بسم الله الرحمن الرحیم

ساختمان داده‌ها

ارزیابی عبارتهای محاسباتی

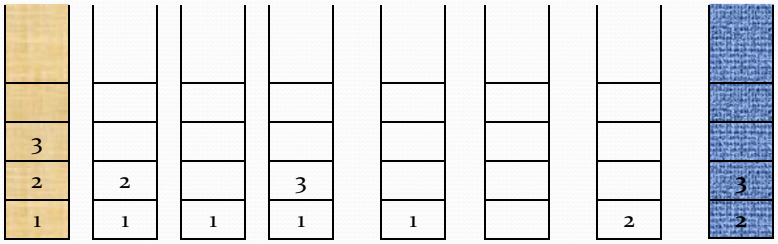
مثال از عملیات روی پشته

- اگر عناصر ۱، ۲ و ۳ به ترتیب در یک پشته قرار بگیرند، وضعیت پشته و متغیرهای (x,y,z) پس از انجام عملیات زیر از چپ به راست چگونه خواهد بود؟

pop(x), pop(y), push(x), pop(z), pop(x), push(y), push(z)

مثال از عملیات روی پشته

- اگر عناصر ۱، ۲ و ۳ به ترتیب در یک پشته قرار بگیرند، وضعیت پشته و متغیرهای (x,y,z) پس از انجام عملیات زیر از چپ به راست چگونه خواهد بود؟



pop(x), pop(y), push(x), pop(z), pop(x), push(y), push(z)

x=3 y=2 x=3 z=3 **x=1** **y=2** **z=3**

عبارتهای محاسباتی

- عبارات محاسباتی:
  - ترکیبی از عملگرها و عملوندها.
- در عبارت محاسباتی a × b،
  - a و b عملوند و × عملگر است.
- عملگرها در یک عبارت،
  - **دارای اولویت مشخص (تقدم) هستند.**
- در صورتیکه تقدم بعضی عملگرها با یکدیگر برابر باشد،
  - در عبارت نسبت به هم تقدم مکانی دارند.
  - یعنی هر کدام زودتر در عبارت ظاهر شود تقدم بیشتری دارد.

اولویت	نام عملگر
۱	() یا پرانتز
۲	- (منفی)، + (مثبت)
۳	↑یا ^ (توان)
۴	* (ضرب)، / (تقسیم)
۵	+ (جمع)، - (تفریق)

داده های نیمه ایستا

عبارتهای محاسباتی-مثال

1. اولویت عملگرهای عبارت زیر را مشخص کنید.

- a / (b – c) – e * (f + g)

2. حاصل عبارت زیر را بدست آورید.

- 14 – 2 * 15 / 3 + 5 ^ 2 ^ 2 – 9

داده های نیمه ایستا

عبارتهای محاسباتی میانوندی و پسوندی

- عبارت میانوندی
  - عملگر بین عملوندها می‌آید. ← حالت متداول
- A/B-C+D*E-A*C
- عبارت پسوندی
  - عملگر پس از عملوندها می‌آید.
  - عبارت پسوندی نیازی به پرانتز ندارد چراکه ترتیب عملگرهایش مشخص است.
- AB/C-DE*+AC*-
- روشهای تبدیل عبارت میانوندی به پسوندی
  - پرانتزگذاری
  - استفاده از پشته
  - پیمایش درخت عبارت محاسباتی

عبارتهای محاسباتی

روشهای تبدیل عبارت میانوندی به پسوندی روش پرانتزگذاری

- عبارت را به طور کامل برحسب اولویت عملگرها پرانتزگذاری کنید و در حین پرانتزگذاری عملگر مربوط به هر پرانتز را روی پرانتز بسته بگذارید.
- تمام پرانتزها را حذف کنید و عبارت را از چپ به راست (شامل عملوندها و عملگرهای روی پرانتزهای بسته) در خروجی بنویسید.
- نکته: در تبدیل عبارات میانوندی به پسوندی، ترتیب عملوندها به هیچ وجه عوض نمی شود، ولی ترتیب عملگرها ممکن است عوض شود.

عبارتهای محاسباتی

روشهای تبدیل عبارت میانوندی به پسوندی روش پرانتزگذاری **مثال**

عبارت میانوندی زیر را به عبارت پسوندی تبدیل کنید.

A/B-C+D*E-A*C

ابتدا پرانتزگذاری انجام می‌شود.

(((A/B)-C)+(D*E)-(A*C))

پرانتزها را حذف می‌کنیم و عبارت را از چپ به راست (شامل عملوندها و عملگرهای روی پرانتزهای بسته) در خروجی می نویسیم.

AB/C-DE*+AC*-

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته

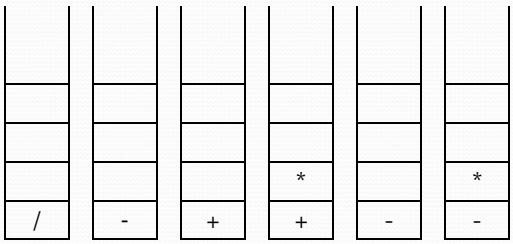
1) عبارت را از سمت چپ پیمایش می کنیم.
2) عملوندها را در خروجی می نویسیم.
3) به هر پرانتز "(" که رسیدیم آن را در پشته قرار می دهیم.
4) به هر عملگر که رسیدیم به شرطی که اولویت آن از عملگر بالای پشته بیشتر باشد، آن را داخل پشته قرار می دهیم، در غیر این صورت آن قدر از بالای پشته عملگر خارج کرده و در خروجی می نویسیم تا یا پشته خالی شده و یا به عملگری برسیم که بتوانیم عملگر مورد نظر را روی آن در پشته قرار دهیم.
5) اگر پرانتز ")" بالای پشته باشد، هر عملگری می تواند روی آن قرار گیرد.
6) به هر پرانتز ")" که رسیدیم آن قدر از پشته عملگر خارج کرده و در خروجی می نویسیم تا به "(" برسیم. در این وضعیت "(" و ")" با هم خنثی می شوند.
7) وقتی به انتهای عبارت رسیدیم، اگر پشته خالی نباشد آن را به طور کامل در خروجی می نویسیم.

نکته: عملگر + و - نمی تواند روی + یا - و عملگر / و * نمی تواند روی / یا * قرار گیرد. ولی ^ می تواند روی ^ قرار گیرد.

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال اول**

- عبارت میانوندی زیر را با استفاده از پشته به عبارت پسوندی تبدیل کنید.

A/B-C+D*E-A*C



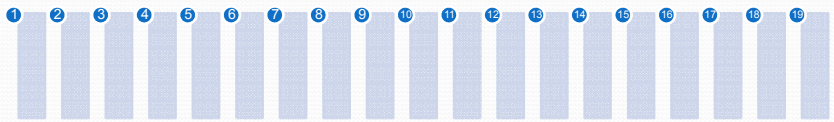
AB/C-DE*+AC*-

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

1

A + (B*C - (D/E^F) * G) *H



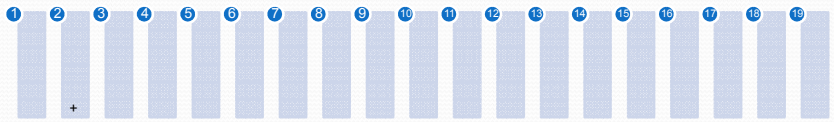
خروجی A :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

2

A + (B*C - (D/E^F) * G) *H



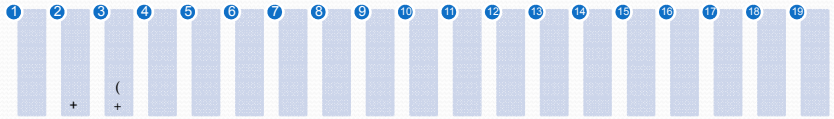
خروجی A :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

3

A + (B*C - (D/E^F) * G) *H



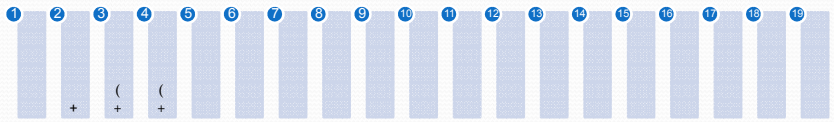
خروجی A :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

4

A + (B*C - (D/E^F) * G) *H



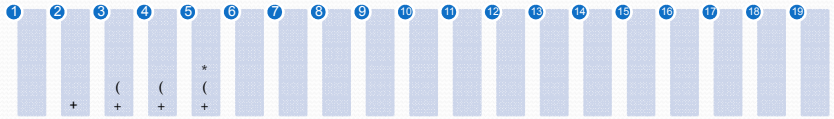
خروجی AB :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

5

A + (B*C - (D/E^F) * G) *H



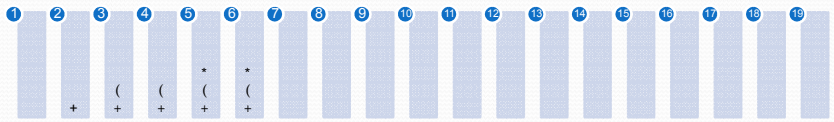
خروجی AB :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

6

A + (B*C - (D/E^F) * G) *H



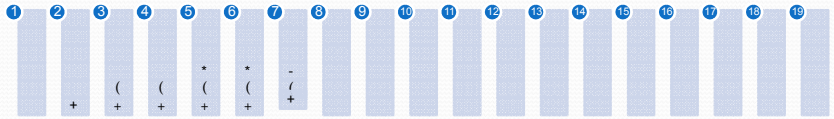
خروجی ABC :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

7

A + (B*C - (D/E^F) * G) *H



خروجی *ABC :

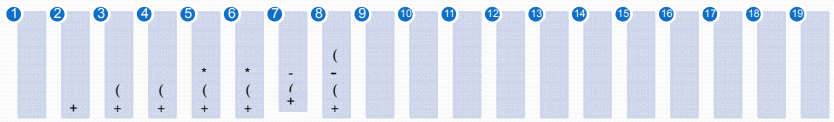
۱۷

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

8

A + (B*C - ((D/E^F) * G) *H



خروجی *ABC :

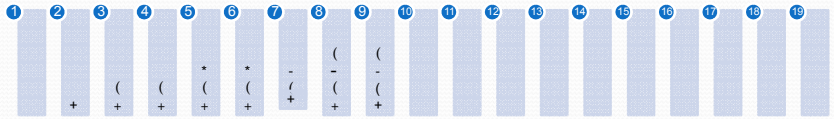
۱۸

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

9

A + (B*C - (D/E^F) * G) *H



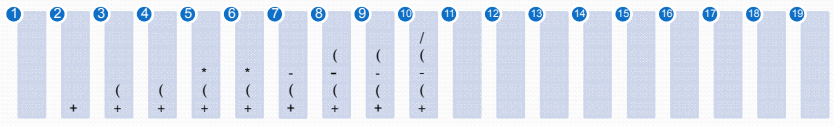
خروجی ABC*D :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

10

A + (B*C - (D/E^F) * G) *H



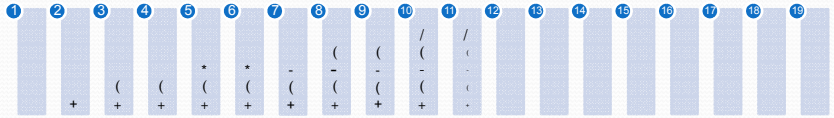
خروجی ABC*D :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

11

A + (B*C - (D/E^F) * G) *H



خروجی ABC*DE :

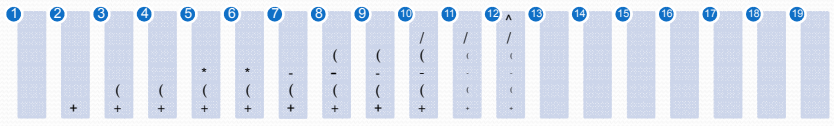
۲۱

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

12

A + (B*C - (D/E^F) * G) *H



خروجی ABC*DE :

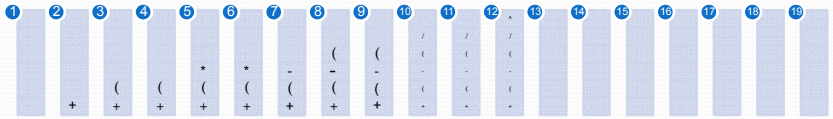
۲۲

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

13

A + (B*C - (D/E^F) * G) *H



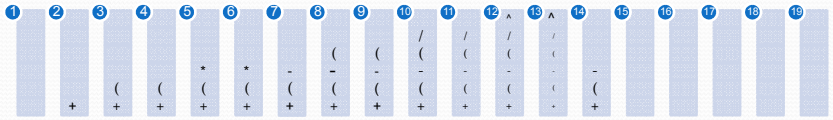
خروجی ABC*DEF :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

14

A + (B*C - (D/E^F) * G) *H



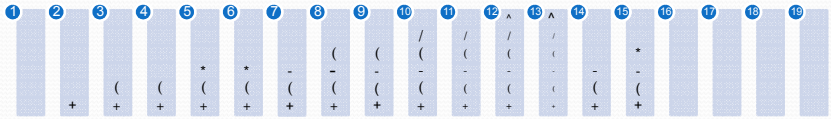
خروجی ABC*DEF^/ :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

15

A + (B*C - (D/E^F) * G) *H



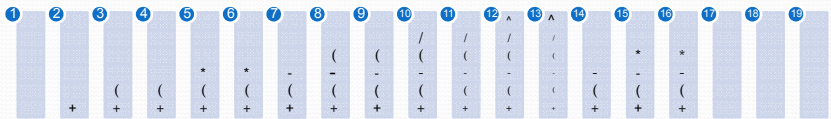
خروجی/ABC*DEF^ :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

16

A + (B*C - (D/E^F) * G) *H



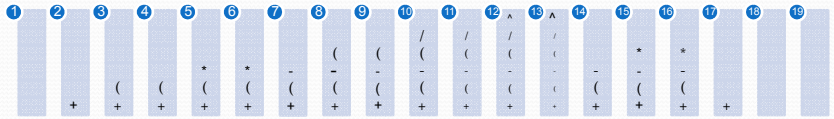
خروجی/G ABC*DEF^ :

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

17

A + (B*C - (D/E^F) * G) *H



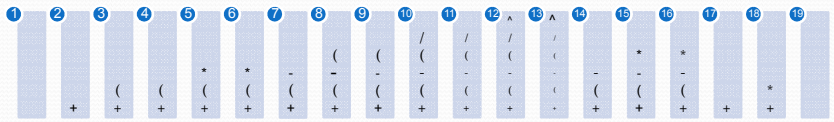
خروجی: ABC*DEF^/G*-

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته مثال دوم

18

A + (B*C - (D/E^F) * G) *H



خروجی: ABC*DEF^/G*-

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

19

A + (B*C - (D/E^F) * G) *H

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	+	(	(	*	*	-	(	(	/	/	^	^	-	*	*	+	*	*
		+	+	(	(	(	-	-	(	(	/	/	(	(	-	-		+	+
				+	+	+	+	+	-	-	(	(	/	/	(	(			
									+	+	-	-	(	(	-	-			
											+	+	-	-	+	+			

خروجی ABC*DEF^/G*-H :

۲۹

داده های نیمه ایستا

عبارتهای محاسباتی روشهای تبدیل عبارت میانوندی به پسوندی با استفاده از پشته **مثال دوم**

A + (B*C - (D/E^F) * G) *H

20

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	+	(	(	*	*	-	(	(	/	/	^	^	-	*	*	+	*	*
		+	+	(	(	(	-	-	(	(	/	/	(	(	-	-		+	+
				+	+	+	+	+	-	-	(	(	/	/	(	(			
									+	+	-	-	(	(	-	-			
											+	+	-	-	+	+			

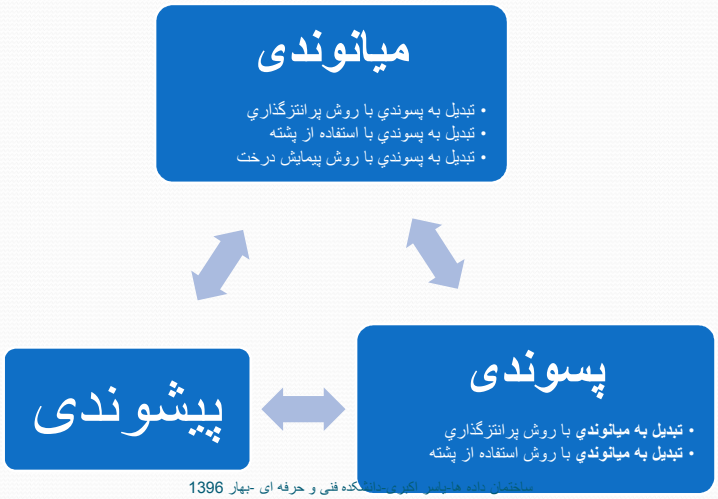
خروجی ABC*DEF^/G*-H*+ :

۳۰

ساختمان داده ها-یاسر اکبری-دانشکده فنی و حرفه ای -بهار 1396

داده های نیمه ایستا

عبارتهای محاسباتی



داده های نیمه ایستا

تبدیل عبارت پسوندی به میانوندی

- روش پرانتزگذاری
  - عبارت را از سمت چپ که با عملوند شروع می‌شود بررسی می‌کنیم.
  - با رسیدن به هر عملگر دو عملوند از سمت چپ برای آن در نظر می‌گیریم.
  - برای دو عملوند، عملگر را بین آن ها قرار داده و برای آن ها پرانتز در نظر می‌گیریم.
  - عبارت را از سمت چپ به راست به ترتیب شامل عملوندها و عملگرهای بین عملوندها در خروجی می‌نویسیم.
- مثال:

6, 2, 3, +, –, 3, 8, 2, /, +, *, 2, ^, 3, +

حاصل عبارت برابر با ۵۲ است.

داده های نیمه ایستا

عبارتهای محاسباتی

تبدیل عبارت پسوندی به میانوندی

- روش استفاده از پشته
  - عبارت را از سمت چپ که با عملوند شروع می‌شود بررسی می‌کنیم.
  - عملوندها را در پشته قرار می‌دهیم و با رسیدن به هر عملگر (به جز عملگر آخر) دو عملوند از پشته خارج کرده و بعد از محاسبه دوباره حاصل عبارت را در پشته قرار می‌دهیم.
  - به عملگر آخر که رسیدیم، بعد از خارج کردن دو عملوند آخر و محاسبه، آن را دیگر در پشته قرار نمی دهیم و حاصل را در خروجی می‌نویسیم.
- مثال:

a, b, c, *, d, /, +

پاسخ برابر با a+((b*c)/d) است.

۳۳

داده های نیمه ایستا

تمرین

- معادل عبارت پسوندی عبارت میانوندی زیر را بدست آورید. همچنین مقدار آنرا مشخص کنید.

5 + 6 ^ 2 / 4 – 15 % 4 * 9 * 2 ^ 3 ^ 2 * 4

- به پشته زیر،۶ خانه در حافظه اختصاص داده شده است.

Top=2 Stack: 5,9,12,-,-,-

خروجی الگوریتم زیر چه خواهد بود؟

```
pop(A)
pop(B)
push(B+2)
push(8)
push(A+B)

Repeat while (top!=-1)
   pop(item)
   Write item
```

۳۴

بسم الله الرحمن الرحیم

ساختمان داده‌ها

لیست پیوندی

داده های برپای خطی

مزایا و معایب آرایه

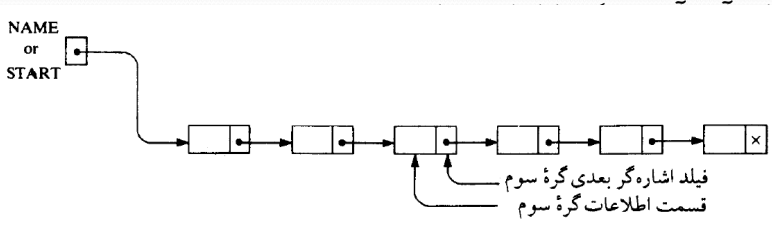
- تعداد عناصر هر آرایه همیشه محدود و مشخص و ایستا است و در تمام طول برنامه ثابت می‌باشد. (عیب)
- عملیات حذف و درج یک داده دلخواه در ناحیه ای مشخص نیاز به شیفت دارد. (عیب)
- روش های مختلفی برای جست و جو در آرایه وجود دارد. (مزیت)
- دسترسی به داده های هر آرایه به صورت تصادفی و دلخواه به راحتی توسط اندیس آرایه انجام می‌شود. (مزیت)
- روش های مختلفی برای مرتب سازی آرایه از قبیل: مرتب سازی حبابی، مرتب سازی سریع، مرتب سازی درجی و ... وجود دارد. (مزیت)

تفاوت آرایه و لیست پیوندی

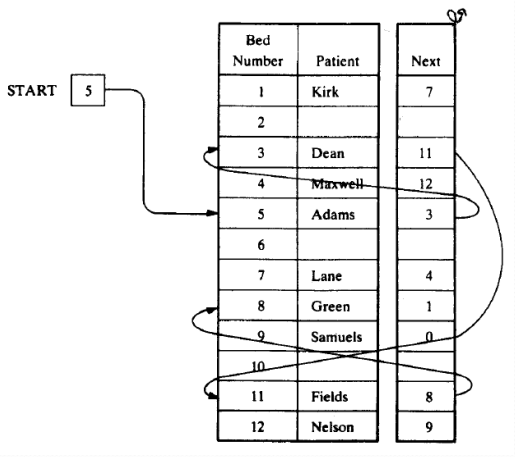
- معایب آرایه
  - داده‌های آرایه ناحیه پیوسته ای از حافظه را در اختیار دارند.
  - ساختمان داده ایستا
  - پرهزینه بودن عملیات حذف و اضافه
- مزایای لیست پیوندی
  - لازم نیست عناصر متوالی داخل لیست، فضای مجاور در حافظه را اشغال کنند.
  - ساختمان داده پویا
  - آسان بودن عملیات حذف و اضافه

لیست پیوندی

- لیست پیوندی مجموعه‌ای خطی از عناصر داده‌ای به نام گره‌هاست که در آن ترتیب خطی توسط اشاره‌گرها داده می‌شود.
- هر عنصر شامل داده و آدرس عنصر بعد (اشاره‌گر یا پیوند) می‌باشد.
- اشاره‌گر آخرین گره، اشاره‌گر Null (پوچ) است.
- لیست پیوندی یک متغیر اشاره‌گر به لیست دارد که نام آن Start است و



قرار گرفتن لیست پیوندی در حافظه



نکاتی در مورد لیست پیوندی

- هر عنصر از لیست پیوندی که به نام گره یا node شناخته می‌شود، حداقل شامل دو فیلد است:
  1. داده (Data)
  2. پیوند یا اشاره‌گر (Link)

Node

data	link

- در حالت خاص که لیست خالی باشد، متغیر Start به Null اشاره می‌کند.

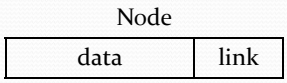
عملیات روی لیست پیوندی

- پیاده‌سازی لیست پیوندی
- پیمایش لیست پیوندی
- جستجو در لیست پیوندی
- اضافه کردن یک گره به لیست پیوندی
- حذف کردن یک گره از لیست پیوندی

پیاده‌سازی لیست پیوندی یکطرفه

- برای تعریف ساختار یک گره از کلاس استفاده می‌شود.

```
class Node{
        int  data;
        Node* link;
}
```



مفهوم اشاره گر

- اشاره‌گر (pointer): یک متغیر است که آدرس متغیرهای دیگر در حافظه را نگهداری می‌کند. اشاره‌گر باید از نوع همان متغیر ولی با * باشد. مثلا متغیری که آدرس یک متغیر int را نگه می‌دارد از نوع int* است.

*i = j
i = &j

- از طریق اشاره‌گر میتوان محتویات محل مورد اشاره را تغییر داد، بنابراین لازم نیست آن محل دارای نام باشد.
- نحوه دسترسی به هر قسمت از محتوای اشاره‌گر (اگر بر فرض مثال، i به یک گره از لیست پیوندی اشاره کند) به صورت زیر است:

*i.data معادل با i->data
*i.link معادل با i->link

مثال پیاده‌سازی لیست پیوندی یکطرفه

- گره p را که به صورت زیر است، ایجاد کنید و سپس حذف کنید.

20	null

(فعلا این گره را خارج از لیست پیوندی در نظر بگیرید.)

- تخصیص فضای حافظه به یک گره با استفاد از new و آزادسازی فضای حافظه آن با استفاده از delete انجام خواهد شد.
- Node* p = new Node()
- p->data = 20
- p->link = null
- Delete p

انواع لیست پیوندی

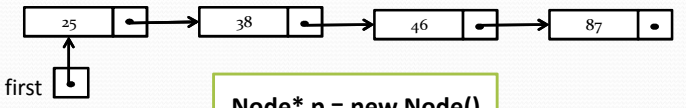
- **لیست پیوندی خطی**
  - لیست پیوندی خطی یک طرفه: در هر گره فقط یک فیلد اشاره‌گر وجود دارد که آن هم آدرس گره بعدی را دارد.
  - لیست پیوندی خطی دو طرفه (مضاعف): در هر گره دو فیلد اشاره‌گر وجود دارد که یکی به گره بعدی و دیگری به گره قبلی اشاره می کند.
- **لیست پیوندی حلقوی**
  - لیست پیوندی حلقوی یک طرفه: اشاره‌گر Link گره آخر به گره اول اشاره می‌کند.
  - لیست پیوندی حلقوی دو طرفه (مضاعف): اشاره‌گر Rlink گره آخر به گره اول و اشاره‌گر LLink گره اول به گره آخر اشاره می‌کند.

عملیات روی لیست پیوندی یکطرفه

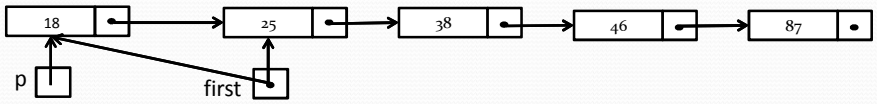
- **اضافه کردن گره به لیست پیوندی**: برای درج هر گره باید از موقعیت قبل از آن، دو عمل جایگزینی را انجام دهیم. شامل چهار حالت است که در دو گروه بررسی می شود:
  - اضافه کردن گره به لیست *پیوندی تهی* یا *ابتدای لیست*
  - اضافه کردن گره به *وسط* یا *انتهای* لیست
- **حذف گره از لیست پیوندی**: برای حذف هر گره باید از موقعیت قبل از آن گره، یک عمل جایگزینی را انجام دهیم. شامل سه حالت است که در دو گروه بررسی می شود.
  - حذف گره از *ابتدای* لیست
  - حذف گره از *وسط* یا *انتهای* لیست

اضافه کردن گره به ابتدای لیست پیوندی

- فرض کنید می خواهیم عدد 18 را به ابتدای لیست زیر اضافه کنیم.

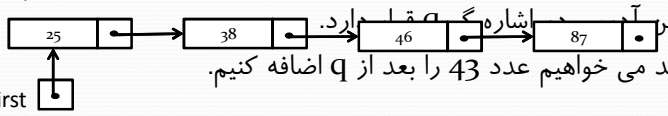



```
Node* p = new Node()
p->data = 18
p->link = first
first = p
```

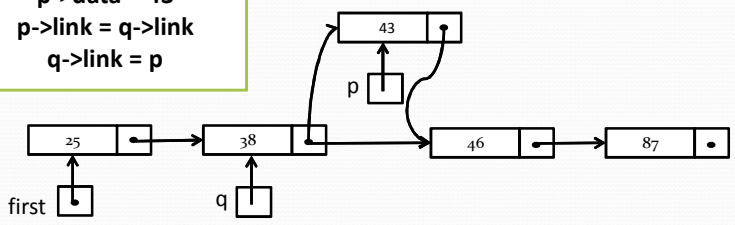


اضافه کردن گره به ابتدای لیست پیوندی

- در این حالت باید آدرس گره قبل از محل درج گره جدید را بدانیم. فرض می‌کنیم این آدرس در اشاره‌گر q قرار دارد.
- فرض کنید می خواهیم عدد 43 را بعد از q اضافه کنیم.

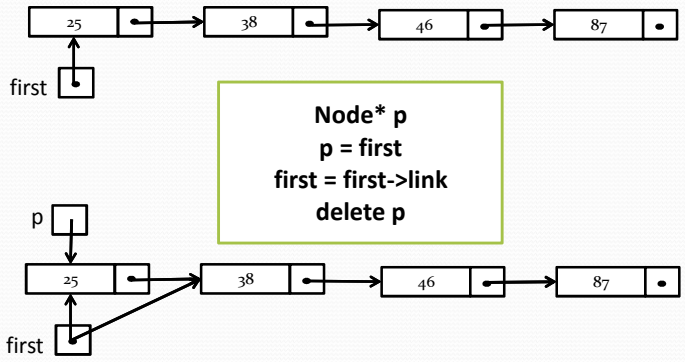



```
Node* p = new Node()
p->data = 43
p->link = q->link
q->link = p
```



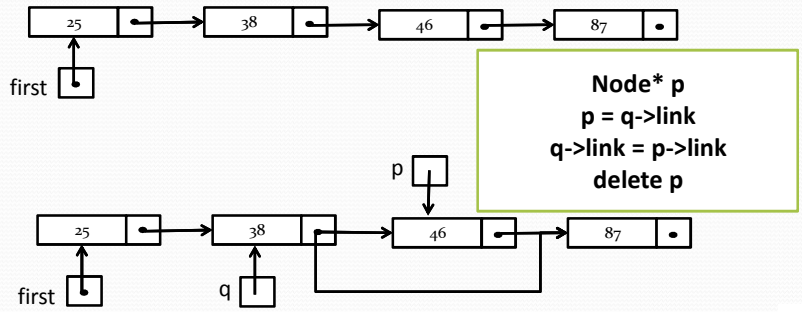
حذف کردن گره به ابتدای لیست پیوندی

- فرض کنید می‌خواهیم گره حاوی مقدار 25 را حذف کنیم.



حذف کردن گره از وسط لیست پیوندی

- در این حالت باید آدرس گره قبل از محل حذف را داشته باشیم. فرض می کنیم این آدرس در اشاره‌گر q قرار دارد.
- فرض کنید می خواهیم گره حاوی مقدار 46 را حذف کنیم.



عملیات روی لیست پیوندی دو طرفه

```
class DNode{
        int  data;
        DNode* left;
         DNode* right;
}
```

DNode

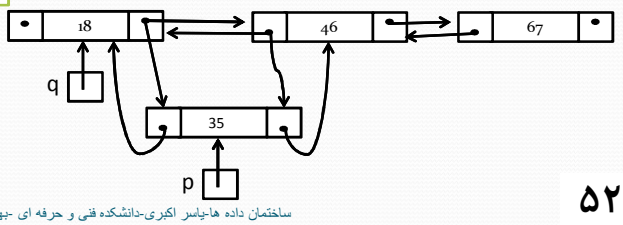
left	data	right

- **اضافه کردن گره به لیست دو پیوندی**: با داشتن آدرس گره دلخواه p در یک لیست دو پیوندی می توان گره جدید را در سمت چپ یا راست گره p درج کرد. برای درج یک گره دلخواه نیاز به چهار عمل جایگزینی داریم.
- **حذف گره از لیست دو پیوندی**: با داشتن آدرس گره دلخواه p در یک لیست دو پیوندی می توان گره مورد نظر را حذف کرد. برای حذف یک گره دلخواه نیاز به ۲ عمل جایگزینی داریم.

اضافه کردن گره به لیست پیوندی دو طرفه

- فرض می کنیم آدرس گره قبل از محل درج در اشاره‌گر q باشد.
- فرض کنید می خواهیم عدد 35 را به لیست اضافه کنیم.

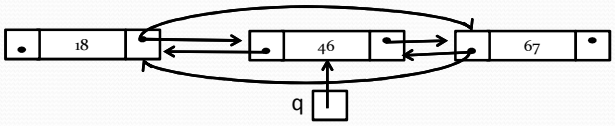
```
DNode* p = new DNode()
p->data = 35
p->left = q
p->right = q->right
q->right -> left = p
q->right = p
```



حذف کردن گره از لیست پیوندی دو طرفه

- فرض می‌کنیم آدرس گره مورد حذف در اشاره‌گر q باشد.
- فرض کنید می‌خواهیم گره حاوی عدد 46 را حذف کنیم.

```
q->left->right = q->right
q->right->left = q->left
delete q
```



بسم الله الرحمن الرحیم

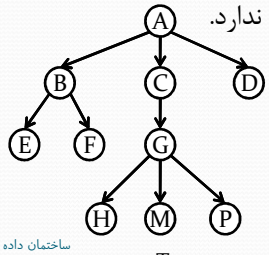
ساختمان داده‌ها

درخت

درخت

- درخت مجموعه ای از یک یا چند گره است طوری که:
  - یکی از گره‌ها ریشه نام دارد.
  - سایر گره‌ها به مجموعه‌های مجزای $T_1...T_n$ تقسیم می‌شوند که هر یک از این مجموعه ها خود، درخت می باشند. $T_1...T_n$ را زیر درختهای ریشه می‌نامند.

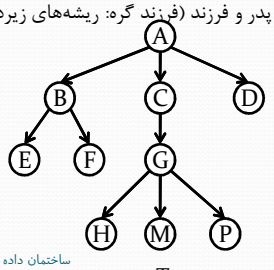
شرط مجزا بودن زیر درختان به این معنی است که هیچ اتصالی بین زیردرختان وجود ندارد.



درخت T

تعاریف مربوط به درخت

- تعداد زیر درختهای یک گره، درجه آن گره نامیده می شود.
- گره با درجه صفر را برگ گویند.
- درجه یک درخت برابر با بیشترین درجه در آن درخت است.
- مفهوم پدر و فرزند (فرزند گره: ریشه‌های زیردرختان آن گره)

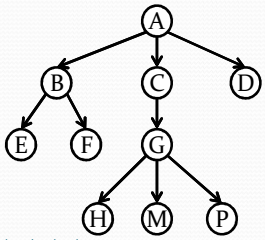


درخت T

داده های چرخشی

تعاریف مربوط به درخت

- مفهوم برادر (همزاد): گره‌های با پدر مشترک
- تمامی گره‌ها در مسیر گره تا ریشه را اجداد (ancestor) آن گره می‌نامند.
- مفهوم سطح: ریشه در سطح یک، فرزندان آن در سطح دو و .... قرار دارند.
- عمق یا ارتفاع یک درخت برابر با حداکثر سطح گره‌ها در آن درخت است.



درخت T

داده های چرخشی

تعاریف مربوط به درخت

- هر خط اتصال هر گره به گره دیگر نامیده می‌شود.
  - در هر درخت دلخواه:
- دنباله ای از یالهای متوالی از یک گره به گره دیگر نامیده می‌شود.
  -
- درختی که تعداد فرزندان هر گره حداکثر k است، نامیده می شود.
- درخت متوازن:
  -
- درخت کاملا متوازن:
  -
- **درخت دودویی (Binary Tree):**
  -
- در درخت دودویی بین فرزند سمت چپ و سمت راست تمایز قائل هستیم.

داده های چندبعدی غیرخطی

تعاریف چند نوع درخت دودویی

- درخت پر:
  - درخت دودویی که همه گره‌های تمام سطوح در آن، ۲ فرزند دارد.
- درخت کامل:
  - درخت دودویی که تمام سطوح به جز احتمالا سطح آخر حداکثر فرزندان را داشته باشد و سطح آخر نیز از سمت چپ گره‌ها پر می‌شود (تا سطح h-1 پر است و گره‌های سطح آخر از چپ به راست چیده شده است)
  - هر درخت پر، کامل و متوازن و کاملا متوازن است.
  - هر درخت کامل متوازن است ولی همیشه پر یا کاملا متوازن نیست.
- درخت دودویی متمایل به راست:
  - هر گره فرزند سمت راست پدر خود است.
- درخت دودویی متمایل به چپ:
  - هر گره فرزند سمت چپ پدر خود است.

داده های چندبعدی غیرخطی

چند نمونه درخت دودویی

- درخت دودویی متمایل به راست:
- درخت دودویی پر:
- درخت دودویی متمایل به چپ:
- درخت دودویی کامل:

نمایش درخت دودویی با آرایه

- برای نمایش درختان دودویی از دو روش آرایه و لیست پیوندی استفاده می‌شود.
- **نمایش آرایه:** در این نمایش با فرض آن که به ریشه در سطح ۱ شماره ۱ بدهیم، بقیه گره‌ها از سمت چپ به راست در سطوح بعدی می‌توانند شماره بگیرند. در واقع اندیس هر گره در آرایه برابر با شماره آن گره در درخت خواهد بود و روابط زیر برای درخت کامل برقرار است.

چند نکته در مورد درخت دودویی

- حداکثر تعداد گره ها در سطح iام یک درخت دودویی 2^{i-1} است.
- حداکثر تعداد کل گره ها در یک درخت دودویی به عمق k برابر با $2^{k}-1$ است.
- حداقل تعداد کل گره ها در یک درخت دودویی به عمق k، k است.
- حداقل تعداد کل گره ها در یک درخت دودویی کامل به عمق k، 2^{k-1} است.
- عمق درخت دودویی پر یا کامل با n گره برابر با $\lfloor \log_2 n \rfloor + 1$ است.

تمرین

- در یک درخت دودویی کامل به عمق ۵ حداقل و حداکثر چند گره وجود دارد؟
- اگر درخت دودویی کاملی ۸ گره داشته باشد، آیا گره شماره ۳ این درخت فرزند چپ و راست دارد؟
- در یک درخت دودویی **کامل** با ۱۰۰۰ گره، عمق چقدر است؟
- در یک درخت دودویی با عمق ۴ حداقل چند گره وجود دارد؟

بسم الله الرحمن الرحیم

ساختمان داده‌ها

دانشگاه جامع علمی-کاربردی

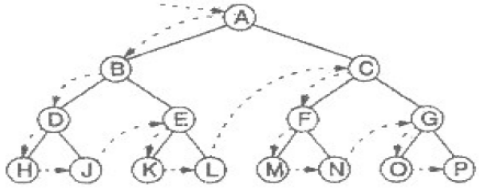
درخت

پیمایش درخت

- پیمایش یعنی ملاقات (visit) همه گره‌های یک درخت بر اساس روشی خاص.
- در پیمایش می‌خواهیم به هر گره فقط یک بار دسترسی داشته باشیم.
- به طور کلی پیمایش درختان در دو حالت و انجام می‌شود.
- برای پیمایش از نمادهای زیر استفاده می‌کنیم:
- در مجموع چهار پیمایش را بررسی می‌کنیم:
  1. پیمایش VLR : PreOrder
  2. پیمایش LVR : InOrder
  3. پیمایش LRV : PostOrder
  4. پیمایش سطحی LevelOrder

پیمایش VLR

- پیمایش **preorder**:
  - ابتدا ریشه درخت ملاقات شده و سپس زیر درخت سمت چپ و پس از آن زیر درخت سمت راست پیمایش می‌شود.
  - هر یک از زیر درختان سمت چپ و راست نیز در صورت وجود به همین روش پیمایش می‌شوند.

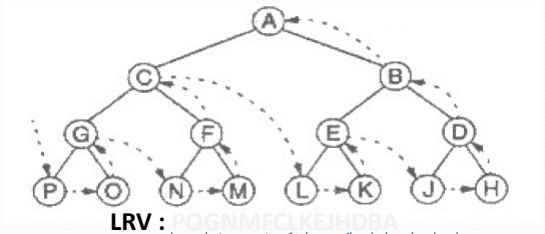


VLR : ABDHJEKLCFMNGOP

داده های چرخشی

پیمایش LRV

- پیمایش postorder:
  - ابتدا زیر درخت سمت **چپ** و سپس زیر درخت سمت **راست** پیمایش شده و پس از آن **ریشه** ملاقات می‌شود.
  - هر یک از زیر درختان سمت چپ و راست نیز در صورت وجود به همین روش پیمایش می‌شوند.

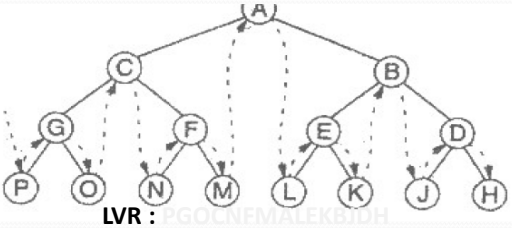


LRV : PGNMFCLKEJHDBA

داده های چرخشی

پیمایش LVR

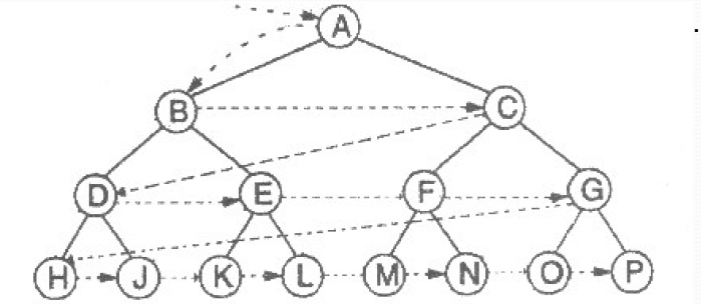
- پیمایش inorder:
  - ابتدا زیر درخت سمت **چپ** پیمایش می‌شود، پس از آن **ریشه** درخت ملاقات شده و سپس زیر درخت سمت **راست** پیمایش می‌شود.
  - هر یک از زیر درختان سمت چپ و راست نیز در صورت وجود به همین روش پیمایش می‌شوند.



LVR : PGCNFMALEKBJDH

پیمایش سطحی

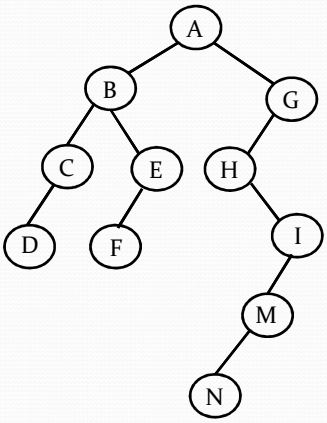
- گره های درخت به ترتیب سطح و چپ به راست ملاقات می‌شوند.



LevelOrder : ABCDEFGHJKLMNOP

پیمایش درخت — مثال

- انواع پیمایش را روی درخت زیر انجام دهید.



LVR: DCBFEAHNMIG

LRV: DCFEBNMIHGA

VLR: ABCDEFGHIMN

LevelOrder : ABGCEHDFIMN

درخت عبارت محاسباتی

- برای تشکیل درخت هر عبارت محاسباتی به صورت زیر عمل می‌کنیم:
  - ابتدا عبارت را برحسب اولویت عملگرها اولویت بندی می‌کنیم.
  - ریشه درخت، کمترین اولویت را دارد.
  - ریشه زیردرختان چپ و راست به ترتیب کم اولویت ترین عملگر در چپ و راست ریشه درخت است.
  - برگهای درخت عملوندها هستند.
- برای تبدیل عبارات میانوندی به پسوندی با استفاده از درخت محاسباتی، از پیمایش postOrder درخت استفاده می‌کنیم.
- برای تبدیل عبارات میانوندی به پیشوندی با استفاده از درخت محاسباتی، از پیمایش preOrder درخت استفاده می‌کنیم.

درخت عبارت محاسباتی

مثال

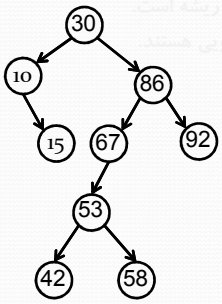
a * b – c / d

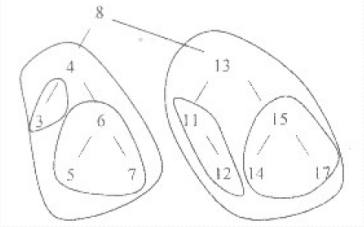
a * b ^ c – d / (e + f)

a – b * c / d ^ (e – f)

درخت جستجوی دودویی BST

- یک درخت دودویی است که می‌تواند تهی باشد و اگر نباشد دارای خصوصیات زیر است:
  1. هر عنصر دارای یک کلید منحصر به فرد است (کلید تکراری نداریم)
  2. مقدار کلید های زیر درخت سمت چپ (در صورت وجود) کمتر از کلید ریشه است.
  3. مقدار کلید های زیر درخت سمت راست (در صورت وجود) بیشتر از ریشه است.
  4. زیر درخت های سمت چپ و سمت راست نیز درختان جستجوی دودویی هستند.





مشخصات مهم درخت جستجوی دودویی

1. هیچ عنصر تکراری در درخت وجود ندارد.
2. پیمایش inorder (LVR) هر درخت BST عناصر را به صورت مرتب شده صعودی نشان می‌دهد.
3. با شروع از ریشه درخت و حرکت به سمت چپ تا رسیدن به گرهی که سمت چپ آن گره تهی است، به گرهی می‌رسیم که مقدار آن در درخت مینیمم است.
4. با شروع از ریشه درخت و حرکت به سمت راست تا رسیدن به گرهی که سمت راست آن گره تهی است، به گرهی می‌رسیم که مقدار آن در درخت ماکزیمم است.

جستجو در درخت جستجوی دودویی

مهمترین عمل در درخت BST جستجو می باشد که از ریشه آغاز شده و حداکثر تا عمق درخت ادامه می‌یابد.

جستجو در داخل درخت جستجوی دودویی به صورت زیر انجام می‌شود:

1. اگر ریشه موجود نباشد جستجو ناموفق است.
2. اگر کلید مورد جستجو با کلید ریشه برابر باشد جستجو به صورت موفق آمیز خاتمه می‌یابد.
3. در غیر این صورت اگر کلید مورد جستجو کوچکتر از ریشه باشد جستجو در زیر درخت سمت چپ و در غیر این صورت در زیر درخت سمت راست به شیوه فوق ادامه می‌یابد.

جستجو در درخت جستجوی دودویی

- حداقل مقایسه،
  - یک مقایسه خواهد بود در صورتی که عنصر مورد جستجو در ریشه باشد.
- حداکثر مقایسه
  - به اندازه عمق درخت خواهد بود در صورتی که عنصرمورد جستجو یکی از عناصر گره‌ها در پایین‌ترین سطح باشد.
- در یک درخت جستجوی دودویی با n گره، به طور متوسط
  - $\log_2 n$ مقایسه انجام می‌شود.
- در یک درخت BST اریب با n گره،
  - حداقل ۱ و حداکثر n مقایسه خواهیم داشت.

کاربرد های درخت جستجوی دودویی

1. جستجو به دنبال داده ها
2. حذف داده های تکراری از لیست ها
3. مرتب سازی (sort) لیست ها با پیمایش inorder (LVR)

درج عنصر در درخت جستجوی دودویی

برای درج یک عنصر ابتدا، عمل جستجو برای گره مورد درج باید انجام شود تا مطمئن شویم گره مورد نظر در درخت BST وجود نداشته باشد، چون در BST عنصر تکراری وجود ندارد. در این وضعیت، دو حالت پیش می‌آید:

1. اگر کلید مورد درج در درخت پیدا شود، در این صورت قابل درج نیست و از لیست ورودی حذف می شود. (به دلیل تکراری بودن)
2. اگر کلید مورد درج در درخت پیدا نشود، در این صورت بر حسب آنکه کوچکتر یا بزرگتر از مقدار کلید آخرین گره مورد جستجو است، به عنوان فرزند سمت چپ یا راست آن درج می‌شود.

دقت کنید برای درج هر گره، مقایسه از ریشه آغاز شده و سعی داریم درخت، حالت BST بودن خود را حفظ کند.

داده های چرخشی

درخت جستجوی دودویی

مثال

- کلیدهای زیر را از چپ به راست در یک درخت جستجوی دودویی درج کنید.

30 , 86 , 67 , 53 , 10 , 58 , 92 , 42 , 15

داده های چرخشی

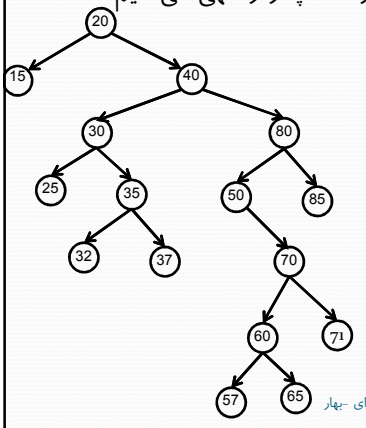
حذف عنصر در درخت جستجوی دودویی

برای حذف عنصر از درخت BST باید ابتدا عمل جستجو انجام گیرد. در این وضعیت دو حالت پیش می‌آید:

- گره مورد نظر برای حذف، در درخت وجود ندارد که در این صورت عملی انجام نمی‌شود.
- گره مورد نظر برای حذف از درخت وجود دارد که در این حالت سه وضعیت بررسی می‌شود:
  1. حذف گره برگ.
  2. حذف گرهی که یک فرزند دارد.
  3. حذف گرهی که دو فرزند دارد.

حذف گره برگ X

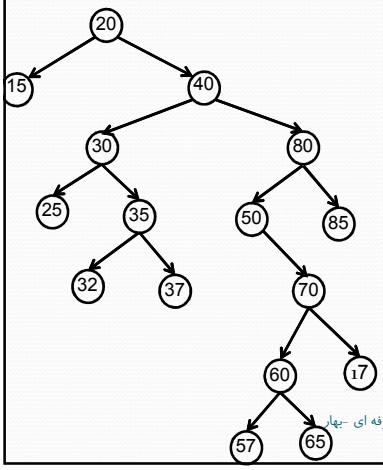
- این حالت بسیار ساده است. اشاره‌گر مناسبی از گره پدر x را تهی می‌کنیم. اگر x فرزند چپ باشد، اشاره‌گر چپ پدر، وگرنه اشاره‌گر راست پدر را تهی می‌کنیم.



حذف 85

حذف گره X با یک فرزند

- گره حذف شده و فرزند جایگزین آن می شود.



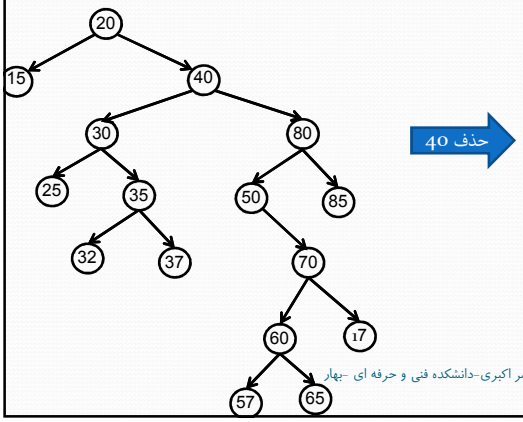
حذف 50

داده های چرخشی

حذف گره X با دو فرزند

به دو صورت ممکن است:

الف) گره حذف و بزرگترین گره زیر درخت سمت چپ جایگزین آن می‌شود.

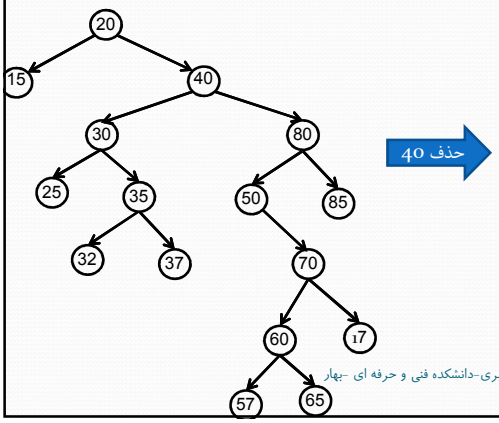


داده های چرخشی

حذف گره X با دو فرزند

ب) گره حذف و کوچکترین گره زیر درخت سمت راست جایگزین آن می‌شود.

- توجه کنید که بزرگ ترین گره سمت چپ و کوچک ترین گره سمت راست گره مورد نظر، فرزند ندارد یا یک فرزند دارد.



داده های چندبای غیرخطی

هرم heap

- Max Tree:
  درختی که مقدار کلید هر گره در آن، بزرگترمساوی مقدار کلید فرزندانش (در صورت وجود) باشد.
- Max Heap:
  یک درخت دودوئی ***کامل*** که Max Tree نیز باشد.
- Min Tree
  درختی که مقدار کلید هر گره در آن، کوچکترمساوی مقدار کلید فرزندانش (در صورت وجود) باشد.
- Min Heap:
  - یک درخت دودویی ***کامل*** که Min Tree نیز باشد.

داده های چندبای غیرخطی

کاربرد هرم heap (برای مطالعه)

- **محاسبه گره مینیمم و ماکزیمم در Heap**
- **درخت minHeap:**

1. عنصر مینیمم در ریشه قرار دارد و بدون مقایسه بدست می‌آید.
2. عنصر ماکزیمم در برگها وجود دارد و چون درخت کامل است تعداد برگها $\lfloor n+1/2 \rfloor$ و تعداد مقایسات برای محاسبه عنصر ماکزیمم $\lfloor n+1/2 \rfloor - 1$ خواهد بود.

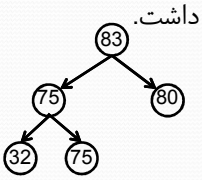
- **درخت maxHeap:**

1. عنصر ماکزیمم در ریشه قرار دارد و بدون مقایسه بدست می‌آید.
2. عنصر مینیمم در برگها وجود دارد و چون درخت کامل است تعداد برگها $\lfloor n+1/2 \rfloor$ و تعداد مقایسات برای محاسبه عنصر ماکزیمم $\lfloor n+1/2 \rfloor - 1$ خواهد بود.

داده های چرخشی

نمایش heap در حافظه

- برای نمایش mintree و maxtree چون لزوما درختان کاملی نیستند از نمایش پیوندی استفاده می‌شود. چون در صورت استفاده از آرایه اتلاف حافظه خواهند داشت.
- برای نمایش maxHeap و minHeap چون درختان کاملی هستند از آرایه استفاده می‌کنیم که در این حالت اتلاف حافظه نخواهیم داشت.



- مثال:
  - نحوه نمایش درخت مقابل در آرایه؟

داده های چرخشی

درج عنصر در maxHeap و minHeap

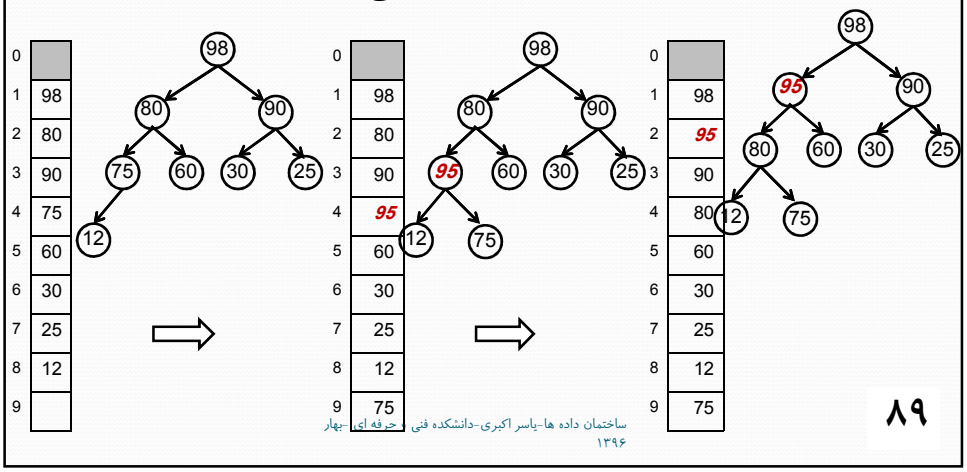
- ابتدا عنصر جدید به گونه‌ای درج می شود که ویژگی کامل بودن درخت حفظ شود. (یعنی در آخرین سطح در اولین موقعیت از چپ به راست)
- سپس درخت به گونه‌ای تنظیم می‌شود تا ویژگی Max Tree بودن آن نیز ایجاد شود. (یعنی در maxHeap تا جاییکه از والدش بزرگتر باشد و در minHeap تا جاییکه از والدش کوچکتر باشد باید با آن جابه‌جا شود.)

داده های چرخشی

درج عنصر در maxHeap و minHeap

مثال

- **مثال ۱**: عنصر 95 را در MaxHeap زیر درج کنید.

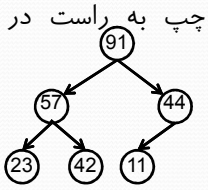


داده های چرخشی

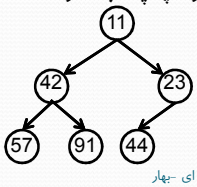
درج عنصر در maxHeap و minHeap

مثال

- مثال ۲:
  - درج عناصر ۴۴، ۹۱، ۵۷، ۱۱، ۴۲، ۲۳ از چپ به راست در درخت maxHeap تهی؟



- مثال ۳:
  - درج عناصر ۴۴، ۹۱، ۵۷، ۱۱، ۴۲، ۲۳ از چپ به راست در درخت minHeap تهی؟



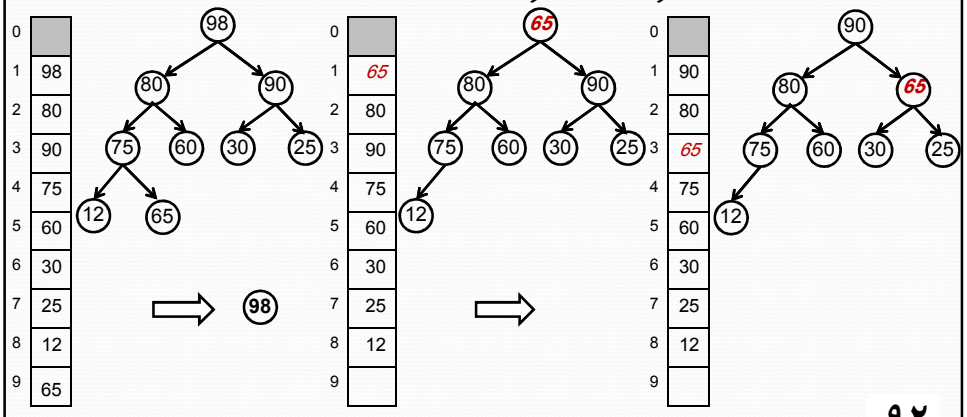
حذف عنصر از maxHeap و minHeap

- حذف از درخت Heap همواره از ***ریشه*** صورت می گیرد.
- پس از حذف ریشه، آخرین عنصر جایگزین آن می‌شود. سپس در maxHeap تا جائیکه از فرزندانش کوچکتر باشد با بزرگترین فرزندش و در minHeap تا جائیکه از فرزندانش بزرگتر باشد با کوچکترین فرزندش جابه‌جا می‌شود.

حذف عنصر از maxHeap و minHeap

مثال

- از MaxHeap زیر یک عنصر حذف کنید.



بسم الله الرحمن الرحیم

ساختمان داده‌ها

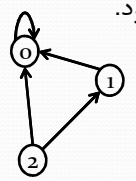
گراف

تعریف گراف

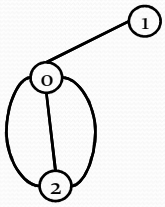
- یک گراف G شامل دو مجموعه V و E است که به صورت G(V, E) نمایش داده می‌شود.
  - V : یک مجموعه محدود و ناتهی از رئوس (Vertices) است.
  - E : یک مجموعه محدود از زوج رئوس که به آنها یال (Edge) می گویند.
- گراف می‌تواند غیر جهت دار یا جهت دار باشد.
  - در گراف غیر جهت دار یال را با زوج (u,v) نمایش می‌دهیم یعنی (u,v) = (v,u).
  - در گراف جهت دار با زوج <u,v> نمایش می‌دهیم یعنی <u,v> ≠ <v,u> .
- **زیرگراف**:
  - هر زیر مجموعه از مجموعه رئوس V و مجموعه یال‌های E را زیر گراف گراف G می‌نامیم

ویژگیهای گراف

- در یک گراف یال‌هایی که ابتدا و انتهای آنها یک راس باشد مجاز نیست. به گرافی که دارای یال حلقه ای باشد گفته می‌شود.



- در گراف یال موازی مجاز نیست. اگر موجود باشد به آن می‌گویند.



تعداد یالها در گراف

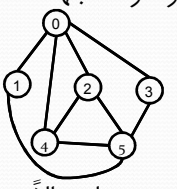
- حداکثر تعداد یالها در گراف **غیر جهت دار** با n راس برابر با
  - n(n-1)/2 می‌باشد.
- حداکثر تعداد یالها در گراف **جهت دار** با n راس برابر با
  - n(n-1) می‌باشد.
- اگر در گراف دلخواه G درجه تمامی رئوس را بدانیم، آنگاه تعداد یالها بر حسب درجه رئوس برابر است با:

تعاریف گراف

- یک گراف غیرجهتدار با n(n-1)/2 یال را گویند.
- در گراف جهت دار درجه یک راس برابر است با:
  - درجه ورودی:
  - درجه خروجی:
- درجه هر گراف برابر با آن گراف است.
- اگر درجه راسی فرد باشد آن راس را و اگر زوج باشد آن راس را گویند.
- **گراف مکمل** گراف G گرافی است که ولی رئوس مشترک دارند.

مفاهیم گراف

- **رئوس مجاور هم (adjacent vertices)**: رئوس واقع شده بر یک یال را گویند.
- **مسیر (path)**: یک مسیر از u به v در گراف G ترتیبی از رئوس $u,i_1,i_2,...i_k,v$ است بطوریکه (u,i_1) , (i_1,i_2) , ... ,(i_k,v) یالهای گراف باشند. اگر گراف جهت دار باشد باید $<u,i_1>$, $<i_1,i_2>$, ... ,$<i_k,v>$ یالهای گراف باشند.
  - مانند : 2,4,5,3,0

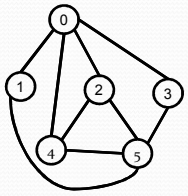


- **طول مسیر**: تعداد یالهای واقع شده بر مسیر را گویند.
  - طول مسیر 2,4,5,3,0 برابر ۴ است.
- **مسیر ساده (simple path)**: مسیری که در آن تمامی رئوس به جز احتمالاً اولین و آخرین آنها مجزا و غیر تکراری است.
  - مانند 2,5,1,0,3

داده های نهای غیرخطی

مفاهیم گراف

- **چرخه (cycle)**: مسیر ساده ای که اولین و آخرین راس در آن یکسان باشد.
  - مانند 2,5,1,0,2
- در گراف بدون جهت دو راس دلخواه u و v را **متصل** می‌گوییم، اگر مسیری از u به v یا بالعکس وجود داشته باشد.
- گراف بدون جهت را متصل یا **همبند** گوییم، اگر برای هر دو راس u و v مسیری از u به v یا بالعکس وجود داشته باشد.



داده های نهای غیرخطی

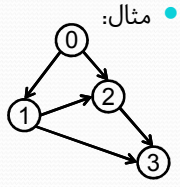
نمایش گراف با استفاده از ماتریس مجاورت

- برای گراف G=(V,E) با n≥1 راس، ماتریس مجاورتی به صورت آرایه دو بعدی زیر تعریف می‌شود.

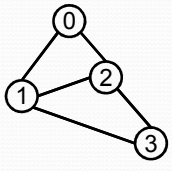
$$A[i][j] = \begin{cases} 1 & \text{اگر یال } (i,j) \text{ یا } <i,j> \text{ موجود باشد} \\ 0 & \text{در غیر این صورت} \end{cases}$$

- مثال:

$$A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



$$A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



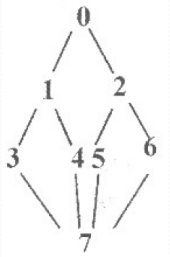
پیمایش گراف

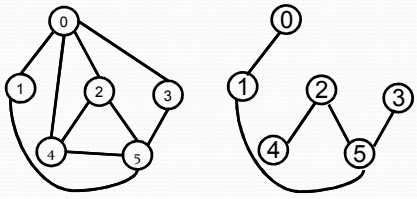
- فرض کنید گراف G=(V,E) و راس v در V را داریم. می‌خواهیم تمامی رئوسی را که در G از طریق v قابل دسترس است را ملاقات کنیم. دو روش وجود دارد:
  - جستجوی عمقی (depth first search)
  - جستجوی سطحی (breadth first search)

- **پیمایش گراف منجر به درخت پوشا می‌شود. درخت پوشای گراف G درختی است که n راس گراف را داشته باشد اما دقیقا n-1 یال را انتخاب کند.**

جستجوی عمقی (DFS)

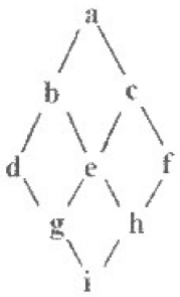
- با راس v شروع کرده سپس یک راس مجاور v که ملاقات نشده را انتخاب و الگوریتم را برای آن تکرار می‌کنیم. وقتی به راسی می‌رسیم که تمامی رئوس مجاورش ملاقات شده، آنگاه به آخرین رأسی برمی‌گردیم که دارای راس مجاور ملاقات نشده باشد.
- مثال: گراف‌های زیر را به روش عمقی پیمایش کنید.

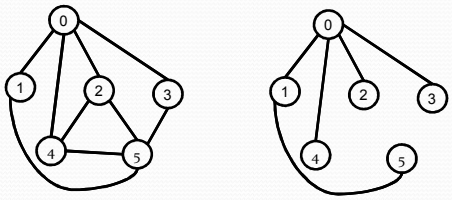




جستجوی سطحی (BFS)

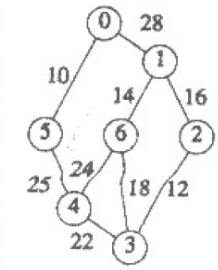
- با راس v شروع کرده سپس تمامی رئوس مجاور v که ملاقات نشده را ملاقات می‌کنیم. پس از آن رئوس ملاقات نشده مجاور با رئوس مجاور v را ملاقات می‌کنیم و تا هنگامیکه همه گره‌ها ملاقات شوند این کار را ادامه می‌دهیم.
- مثال: گراف‌های مقابل را به روش سطحی پیمایش کنید.

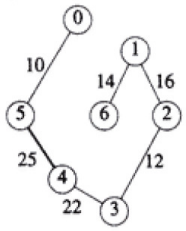




درخت پوشای کم هزینه

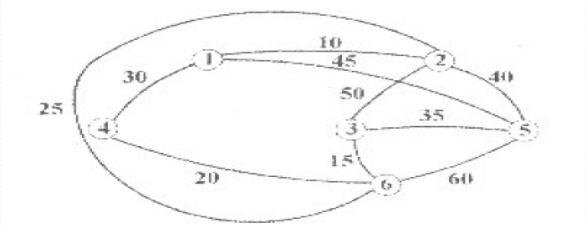
- گرافی که در آن یال‌ها دارای وزن باشند، **گراف وزن‌دار** نامیده می‌شود.
- برای بدست آوردن درخت پوشایی که دارای کمترین هزینه باشد از الگوریتم‌های خاصی استفاده می‌شود، که تمام این الگوریتم‌ها بر دو اصل پیاده سازی می‌شوند:
  - **همیشه در هر مرحله یالی را انتخاب می‌کنیم که کمترین وزن را با مسیر انتخاب شده تا به حال داشته باشد.**
  - **همیشه یالی انتخاب می‌کنیم که با مسیر انتخاب شده تا به حال، دور ایجاد نکند.**





الگوریتم کروسکال

- در الگوریتم کروسکال (Kruskal) یال‌های گراف را به ترتیب صعودی مرتب می‌کنیم. از اولین (کوچکترین) یال شروع کرده و هر یال را به گراف اضافه می کنیم به شرط اینکه دور در گراف ایجاد نگردد. این روال را آنقدر ادامه می دهیم تا درخت پوشای بهینه تشکیل گردد.
- در هر مرحله از الگوریتم کروسکال ممکن است درختها متصل نباشند.
- مثال: درخت پوشای مینیمم گراف مقابل را با استفاده از الگوریتم کروسکال بدست آورید.



۱۰۵

الگوریتم پریم

- در روش پریم (prim) از یک رأس شروع می کنیم و یال با کمترین وزن که از آن می گذرد را انتخاب می کنیم. در مرحله بعد یالی انتخاب می‌شود که کمترین وزن را در بین یال‌هایی که از دو گره موجود می‌گذرد داشته باشیم. این روال را آنقدر تکرار می کنیم تا درخت پوشای بهینه حاصل شود. باید توجه کرد که یال انتخابی در هر مرحله در صورتی انتخاب می‌شود که در گراف دور ایجاد نکند.
- تفاوت روش پریم با روش کراسکال در این است که گراف حاصل در مراحل میانی روش پریم همیشه متصل است ولی در الگوریتم کراسکال در آخرین مرحله قطعاً متصل است.

۱۰۶

کار عملی (ایجاد-اضافه-حذف-پیمایش)

1. آرایه
   - (پیمایش-اضافه کردن یک عنصر-حذف کردن یک عنصر)
2. پشته
3. صف خطی
4. صف حلقوی
5. لیست پیوندی
   - (پیاده‌سازي -پیمایش -جستجو -اضافه کردن یک گره -حذف کردن یک گره)
6. لیست دوپیوندی
   - (پیاده‌سازي -پیمایش -جستجو -اضافه کردن یک گره -حذف کردن یک گره)