بسمه تعالی

ساختمان داده‌ها و دریا سکال

طرح درس

الف- سرفصل

۱. مقدمات

۲. آرایه‌ها

۳. صف و پشته

۴. لیست‌های پیوندی

۵. درخت‌ها و گراف‌ها

*۶. روش‌های مرتب‌سازی

ب- منابع

*۱. ساختمان داده‌ها و دریا سکال (هورویتز) ← صفحاتی که برنامه دارد و الگوریتم‌ها

۲. اصول ساختمان داده‌ها (لیپ شوتز)

۳. ساختمان داده‌ها (جلد شمی اصل)

*۴. جزوهٔ کلاسی

ج- وضعیت نمره

۱. تمرین و کار کلاسی و پروژه (۲۵٪: ۵ نمره) Print

۲. میان ترم (۳۰٪ = ۶ نمره) ← تا فصل ۴ اوایل آذرماه ۱۲۵ آذرماه

۴. پایان ترم (۴۵٪ = ۹ نمره)

۲ پروژه (پروژهٔ اول ۱ نمره پروژهٔ دوم روز امتحان = با Print)

استاد: دلشاد رسیر

terms[a].exp < terms[b].exp (c

در این حالت چون که توان جمله‌ی چند جمله‌ای B(x) بزرگتر است آن جمله B(x) را در C(x) می‌نویسیم و سپس b را یک واحد پیش می‌بریم.

عملیات فوق را آنقدر ادامه می‌دهیم تا یکی از چند جمله‌ای‌های A(x) یا B(x) تمام شود، A(x) زمانی تمام می‌شود که a > FinishA باشد به همین ترتیب B(x) زمانی تمام می‌شود که b > FinishB شود هنگامی که یکی از چند جمله‌ای‌ها تمام شد عیناً جملات چند جمله‌ای دیگر را در C(x) می‌نویسیم. در نهایت برای آنکه پایان چند جمله‌ای C(x) مشخص شود قرار دهیم ; FinishC := Free-1

زیر برنامه‌ی Padd در ص ۶۴ به همراه زیر برنامه‌ی NewTerm در ص ۶۵ عملیات جمع دو چند جمله‌ای را که به صورت فشرده ذخیره شده‌اند انجام می‌دهد:

2 a و b را جمع می‌کند و c را بدست می‌آورد.

3 متغیرها را معرفی می‌کند. A B X

5 خط مقداردهی آغازین INITIALIZE

6 هنوز A(x) و B(x) تمام نشده.

9 مجموع ضرایب در c می‌گذاریم.

10 پروسیجر NEW صدا زده می‌شود.

* NewTerm روتین ورودی می‌گیرد C ← coef / e ← exp

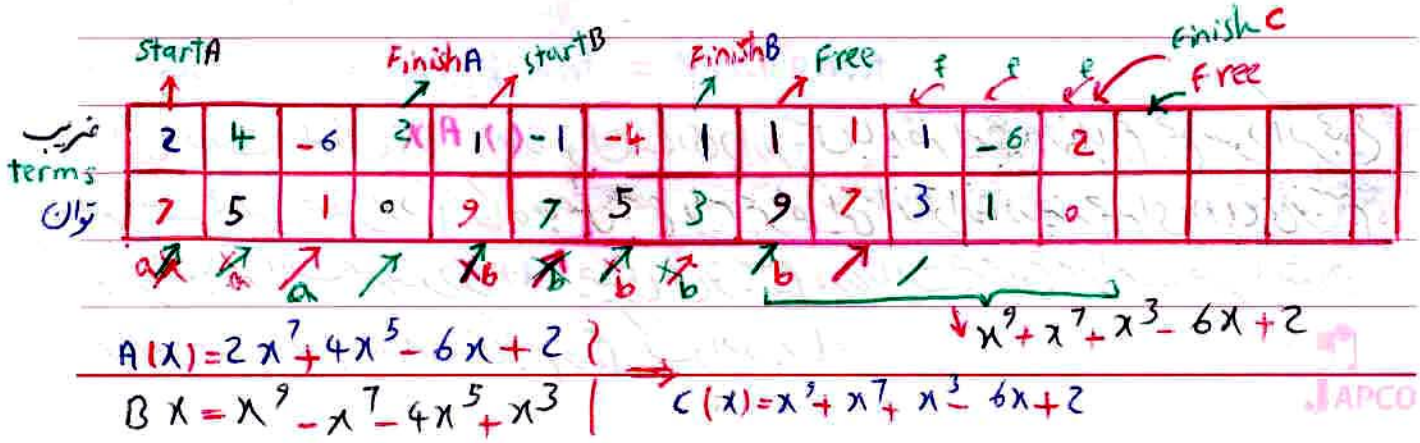
MaxTerms طول آرایه Terms

22 جملات باقیمانده از A(x) را بنویس.

23 تا 27 جملات A را می‌نویسد تا زمانی که برسد به FinishA

29 تا 33 جملات b را می‌نویسد تا زمانی که به FinishB برسد.

34 پایان c را مشخص می‌کند.

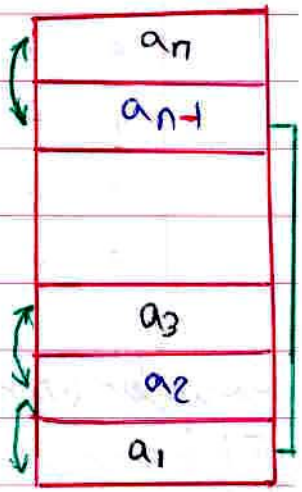


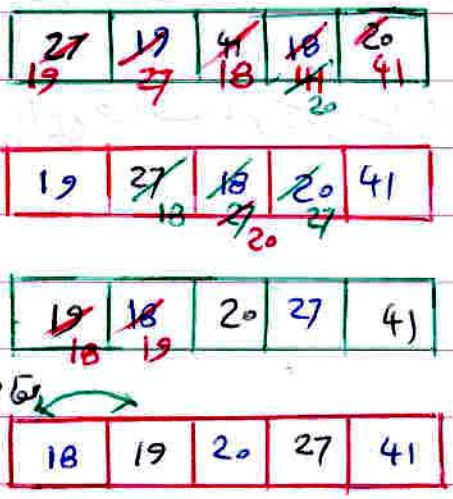
روش مرتب سازی حبابی BUBBLE SORT

در این روش ابتدا در آرایه‌ای به طول n بزرگترین عنصر را به سمت انتهای آرایه ببل می‌دهیم، سپس در آرایه‌ای به طول (n-1) همین کار را تکرار می‌کنیم، در تکرار بعد در آرایه‌ای به طول (n-2) این کار را می‌کنیم و مرتباً آن را تکرار کرده تا در آرایه‌ای به طول 2 عنصر بزرگتر را به سمت خانه دوم ببل می‌دهیم، سپس یک حلقه با متغیر i نیاز داریم که از n تا 2 یکی یکی کم می‌شود، یک حلقه با متغیر j نیاز داریم که از 1 تا (i-1) یکی یکی اضافه می‌شود.

قطعه برنامه زیر عمل مرتب سازی حبابی را انجام می‌دهد:

```
FOR i:= n  DOWNTO 2  DO
  FOR j:= 1  TO  i-1  DO
    IF a[j] > a[j+1]  THEN
      BEGIN
        temp:= a[j];
        a[j]:= a[j+1];
        a[j+1]:= temp;
      END;
```





مقایسه می‌شوند ولی جایجا نمی‌شوند.

زمان اجرا

$$\sum_{i=n}^{2}\sum_{j=1}^{i-1} 3 = 3\sum_{i=n}^{2}\sum_{j=1}^{i-1} 1 = 3\sum_{i=n}^{2}(i-1) = 3\left(\frac{n(n+1)}{2}-1\right) = \frac{3}{2}\left(n(n+1)-2\right)$$

$$\Rightarrow O(n^2),\ k=\frac{3}{2}$$

ماتریس‌های اسپارس (تنک) SPARSE MATRIX

هر ماتریسی را که تعداد صفرهای آن زیاد باشد ماتریس اسپارس می‌گویند، معمولاً ماتریس‌هایی که بیش از نیمی از عناصر آنها صفر باشد جزو ماتریس‌های اسپارس در نظر گرفته می‌شود. اما در عمل اگر در حدود ۸۰ درصد عناصر یک ماتریس صفر باشند خاصیت اسپارس بودن ماتریس به صورت بازه مشخص می‌شود. مثلاً ماتریس زیر یک ماتریس اسپارس است:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 2 & 7 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}_{6\times 7}$$

همانطور که مشاهده می‌شود در این ماتریس تر دیک به ۸۰ درصد عناصر صفر هستند بنابراین به نظر می‌رسد که عناصر صفر را ذخیره نکنیم و تنها عناصر غیر صفر را ذخیره کنیم، پس باید علاوه بر مقدار یک عنصر غیر صفر حتماً محل آن یعنی سطر و ستون را نیز ذخیره کنیم پس برای هر عنصر غیر صفر باید یک سه تایی به صورت زیر در نظر بگیریم

سطر	ستون	مقدار
Row	Column	Value

با این حساب ماتریس اسپارس A در مثال قبل می‌تواند به صورت زیر ذخیره شود:

فشرده شده A: C_A

1	5	3
2	1	-2
4	1	4
4	5	-1
5	2	3
5	5	2
5	6	7
6	4	2

برای آنکه این ساختار سه‌تایی را در زبان پاسکال تعریف کنیم باید در قسمت نوع داده (TYPE) با استفاده از دستور رکورد RECORD نوع داده matrixterm را به صورت زیر تعریف کنیم:

```
TYPE  matrixterm = RECORD
        row : 1 . . MAXINT;
        col : 1 . . MAXINT;
        val : REAL;
      END;
```

و در نهایت در قسمت متغیرها (VAR) می‌توانیم ماتریس اسپارس CA را یک آرایه از جنس ماتریس Term تعریف کنیم:

```
VAR   Ca : ARRAY [0 . . MAXINT] of matrixterm;
```

تذکر: برای آنکه ابعاد ماتریس A در نظر گرفته شود خانه صفر Ca را برای این منظور در نظر گرفتیم، در خانه صفر Ca به ترتیب تعداد سطرها، تعداد ستون‌ها و تعداد عناصر غیر صفر ماتریس A را قرار می‌دهیم، پس عملاً Ca به صورت زیر در نظر گرفته می‌شود:

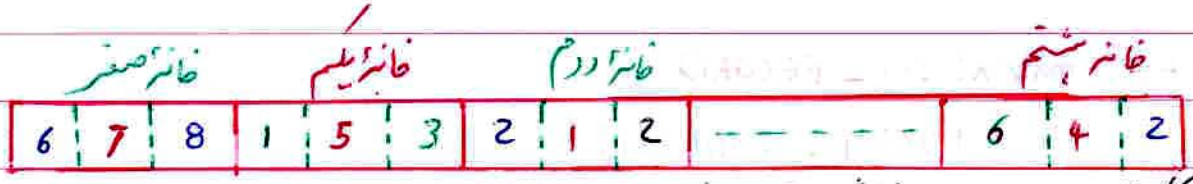
تعداد عناصر غیر صفر = تعداد ستون‌ها = تعداد سطر = n

خانه صفر Ca	6	7	8
Ca:	1	5	3
	2	1	-2
	4	1	4
	4	5	-1
	5	2	3
	5	5	2
	5	6	7
	6	4	2

ادامه

تذکر مهم

همانطور که در تعریف a مشخص شده آرایه a یک بعدی است یعنی به صورت زیری باشد:



برای سادگی کار به این صورت نمایش داده می‌شود.

«مبحث جدید»

ترانهاده کردن ماتریس

۹، ۲، ۲۶

یکی از ساده‌ترین عملیاتی که روی ماتریس‌ها انجام می‌شود پیدا کردن ترانهاده آنهاست، همانطور که از ریاضیات مقدماتی می‌دانیم در ترانهاده ماتریس جای سطر و ستون‌های ماتریس عوض می‌شود یعنی اگر $A=[a_{ij}]_{m\times n}$ یک ماتریس با m سطر و n ستون باشد آنگاه ترانهاده آن که با نماد A^T یا A' نشان داده می‌شود ماتریسی $n\times m$ به صورت زیر خواهد بود:

$$A^T=[a_{ji}]_{n\times m}$$

برای ماتریس‌های معمولی قطعه برنامه ساده زیر عمل ترانهاده را انجام می‌دهد:

```
FOR i := 1 TO m DO
FOR j := 1 TO n DO
    at[j,i] := a[i,j];
```

زمان اجرای این الگوریتم $O(mn)$ است (برحسب درایه‌های ماتریس A)

حال ببینیم برای پیدا کردن ترانهاده یک ماتریس اسپارس که به صورت فشرده ذخیره شده است چگونه می‌توان عمل کرد:

ماتریس مثال قبل را در نظر می‌گیریم:

ادامه ←

$$A=\begin{bmatrix} 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 7 & 0 \\ 0 & 3 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{فشرده}}$$

C_A:

6	7	8
1	5	3
2	1	-2
4	1	4
4	5	-1
5	2	3
5	5	2
5	6	7
6	4	2

بنظرمی رسد برای پیداکردن ترانهادهٔ A ازروی شکل فشردهٔ آن یعنی C_A ساده‌ترین راه آن است که جای ستونهای اول ودوم C_A راعوض کنیم، با این کار عملاً جای سطر و ستون عناصر غیرصفر عوض می شود و داریم:

$(C_A)^T$:

6	7	8
5	1	3
1	2	-2
1	4	4
5	4	-1
2	5	3
5	5	2
6	5	7
4	6	2

$$A^T=\begin{bmatrix} 0 & -2 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 3 & 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{7\times 6}$$

C_{A^T}:

6	7	8
1	2	-2
1	4	4
2	5	3
4	6	2
5	1	3
5	4	-1
5	5	2
6	5	7

اما اگر ابتدا ماتریس A راترانهاده کنیم، سپس ازروی ماتریس ترانهاده یعنی A^T شکل فشردهٔ آن را بنویسیم درنهایت C_{A^T} به صورت زیر بدست می آید:

همانطور که مشاهده می شود سطرهای $(C_A)^T$ و C_{AT} باهم یکی نیستند، در شکل فشرده ماتریس اسپارس عناصر غیرصفر به ترتیب شماره سطر مرتب شده اند ضمناً در آنهایی که شماره سطر یکسان دارند براساس شماره ستون مرتب شده اند، می بینیم $(C_A)^T$ چنین خاصیتی ندارد، اولین ایده که به ذهن می رسد آن است که در $(C_A)^T$ عناصر را مرتب کنیم، اگر ماتریس A، t عنصر غیرصفر داشته باشد، زمان مورد نیاز برای تعویض ستون اول و دوم ماتریس فشرده C_A ← $O(t)$ خواهد بود، پس برای مرتب کردن اگر از یک روش مرتب سازی معمولی مانند مرتب سازی حبابی استفاده کنیم زمان مورد نیاز برای آن نیز $O(t^2)$ خواهد بود و در نهایت زمان این روش:

$$O(t + t^2) = O(t^2)$$

$O(t + t^2)$ ← برای جابجایی ستونهای اول و دوم C_A

← برای مرتب کردن

اگر ماتریس A یک ماتریس $m \times n$ باشد در بدترین حالت $t = mn$ عنصر غیرصفر دارد و در اینجا زمان مورد نیاز الگوریتم $O(m^2n^2)$ خواهد بود که بسیار بدتر از الگوریتم ترانهاده معمولی است که زمان آن $O(mn)$ بود.

روش بهتر برای ترانهاده کردن

الگوریتم ترانهاده TRANSPOSE

در روش قبل دیدیم زمان اجرا بسیار زیاد است، در این قسمت الگوریتم ترانهاده معمولی را معرفی می کنیم، در این الگوریتم به ستون دوم ماتریس C_A توجه می کنیم، در واقع در ستون دوم C_A شماره ستون عناصر غیرصفر A قرار دارد که قرار است در ترانهاده به شماره سطر تبدیل شود، در ستون دوم ماتریس C_A، t عنصر داریم (یادآوری می شود که t تعداد عناصر غیرصفر است) بار اول کلیه اعداد یک را به ماتریس جدید که قرار است ترانهاده A باشد منتقل می کنیم، بار دوم عناصری که شماره Z دارند منتقل می کنیم و همین طور بار nام عناصری که شماره n دارند منتقل می کنیم (n تعداد ستونهای ماتریس A است)

الگوریتم ترانهاده در صفحه ۷۰ آمده است:

b.2 برابر با ترانهاده a قرار داده می شود.

terms .3 همان تعداد عناصر غیرصفر c، تعداد ستونهای ماتریس A، i یک متغیر و شمارنده،

currentB ← B جاری، در جریان ← روی خانه های B حرکت کند.

INITIALIZE : مقداردهی آغازین 8 تا 5

terms = b[0].value 5

ادامه

و، شرط : اگر ماتریس، ماتریس صفر باشد : terms = 0 برنامه کارش تمام شده است ،

(ماتریس 8x13 : | 8 | 13 | 0 | ترانهاده | 13 | 8 | 0 |

اگر غیر صفر باشد مراحل زیر انجام می شود :

12. دفعهٔ اول c = 1

13. i از 1 تا a تغییر می کند

14. عناصر در ستون c را می یابد.

15. داخل این شرط زمانی می آییم a[i] = c یا col برابر 1 است.

زمان اجرای الگوریتم ترانهاده معمولی برابر O(nt) است که در آن n تعداد ستونهای ماتریس A است و t تعداد عناصر غیر صفر A می باشد.

O(tn) → در بدترین حالت O(mn²)

row	col	value
7	6	8
1	2	-2
1	4	4
2	5	3
4	6	2
5	1	3
5	4	-1
5	5	2
6	5	7

وضوع پرورژه

زمان تحویل 13، 8، 81

ضرب چند جمله ای ها

هدف نوشتن برنامه ای است که دو چند جمله ای A(x) و B(x) را اگر به صورت فشرده ذخیره شده اند در هم ضرب کند و حاصل را در چند جمله ای فشرده C(x) قرار دهد. (تذکر: در C(x) نیز ترتیب جملات ذخیره شده. بلکه هم صورت ترتیب نزولی توان باشد.)

راهنمایی :

می توانیم از یک آرایهٔ کلی به نام temp استفاده کنید.

تمرین صفحه ۶۵

۲. چنانچه $A=(a_1,\dots,a_n)$ و $B=(b_1,\dots,b_m)$ هرکدام یک لیست ترتیبی باشند، آنگاه $A<B$ خواهد بود اگر برای $i<j$ ای $1\le i\le n$ و $a_i=b_i$ و $a_j<b_j$ باشد و یا برای $1\le i\le n$ و $n<m$ و $a_1=b_1$ باشد. روالی (زیر برنامه) بنویسید که برحسب اینکه $A<B$ و $A=B$ یا $A>B$ باشد به ترتیب مقادیر ۱-، ۰، ۱+ را برگرداند. فرض نمایید امکان مقایسه مقادیر داده شده a_i و b_j وجود دارد.

a | 12 | 5 | 7 | 3 | $n=4$ | A | 3 | 7 | 4 | $A>B$

$n<m$

b | 12 | 5 | 6 | 4 | 2 | $m=5$ | B | 3 | 7 | 4 | 6 | $B>A$ ← اگر به

```
i:=1;
while ((a[i]=b[i]) AND (i<=m) AND (i<=n)) DO
  i:=i+1;
IF (n=m) AND (i=n+1) THEN F:=0; (writeln('A=B');
IF ((i=n+1) AND (i<=m)) THEN F:=-1;   B>A
IF ((i=m+1) AND (i<=n)) THEN F:=1;    A>B
IF ((i<=n) AND (i<=m)) THEN
  IF a[i]<b[i] THEN F:=-1 ELSE F:=1;
```

آرایه A تمام شده — آرایه B تمام شده — هیچ آرایه‌ای تمام نشده

(در انتهای برنامه F را برمی‌گرداند)

به عنوان تمرین با while

(مهم ترین الگوریتم)

FAST TRANSPOSE ترانهاده سریع (تند)

در روش ترانهاده معمولی زمان اجرا $O(nt)$ شد و در حالتی که $t=mn$ باشد (بدترین حالت) زمان مناسبی نیست، در این قسمت با روشی سریعتر آشنا می شویم. در این روش از 2 آرایه کمکی Row size و Row start کمک می گیریم که طول هردوی آنها n است (به تعداد ستونهای ماتریس A) در واقع Row size (i) تعداد عناصر در ستون iام A (یا تعداد عناصر در سطر iام B) می باشد و Row start (i) را اگر در واقع مکان شروع سطر iام در ماتریس ترانهاده B می باشد به صورت زیر محاسبه می نیم:

```
Row start (1) := 1;
Row start (i) := Row start (i-1) + Row size (i-1);
```

نقش کلیدی را در الگوریتم ترانهاده سریع آرایه Row start برعهده دارد زیرا در واقع مکان شروع را در ماتریس ترانهاده مشخص می کند، با استفاده از این آرایه می توانیم تمام عناصر غیر صفر A را به B منتقل نیم

زیر برنامه الگوریتم ترانهاده سریع در صفحه ۷۲ آمده است:

۱. یک متغیر a از جنس اسپارس و b هم همین طور.
۲. ترانهاده a در b قرار داده می شود ضمناً زمان اجرا $O(n+t)$ است که t تعداد عناصر غیر صفر و n تعداد ستونها می باشد.
۳. تعریف دو آرایه
۶ و ۷. مقدار دهی آغازین
۸. terms کار تمام
۱۰. مقدار Rowsize (i) را اگر برابر تعداد عناصر در سطر iام b می باشد محاسبه می کند.
۱۱. همه را صفر قرار می دهد و مقدار دهی آغازین
۱۲. محاسبات انجام می دهد.

Rowsize (a[i].col)
عدد 2 ← خانه 2

۱۳. توضیح Row start (i) را اگر مکان شروع سطر iام در ماتریس ترانهاده می باشد مشخص می کند (خط ۱۲ و ۱۵)
16 تا 23: انتقال از ماتریس اصلی a به ماتریس ترانهاده b
۲۲. ضمناً یکی از ترایش پیدا کند جمله ن اعداد بعدی را روی این نوشت.

زمان اجرا $O(n+t+n+t) \Rightarrow O(n+t)$

ساختمان داده

هر ساختمان داده یک سه تایی به صورت (D , F , R) می باشد که در آن D مجموعهٔ داده (Domain)، F مجموعهٔ توابع (Functions) و R مجموعهٔ قوانین حاکم بر ساختمان داده است. (Rules)

به عنوان مثال

اگر بخواهیم ساختمان دادهٔ صفهای یک فروشگاه زنجیره ای را در نظر بگیریم مجموعهٔ D به صورت زیر تعریف می شود.

D = { صف لوازم آرایشی ، صف پوشاک ، . . . }

حداقل توابعی که در این ساختمان داده به کار گرفته می شود عبارتند از:

۱. تابع CREAT برای ایجاد صف
۲. تابع ISFULL برای تشخیص پر بودن صف
۳. تابع ADD برای اضافه کردن یک فرد جدید به صف
۴. تابع IS EMPTY جهت تشخیص خالی بودن صف
۵. تابع DELETE برای خارج کردن یک نفر از صف

دامنه و برد این توابع به صورت زیر تعیین می گردد:

CREAT (name , max size) :: = Queue

Is FULL (name , maxsize) :: = Boolean

ADD (item , queue) :: = Queue

IS EMPTY (name) :: = Boolean

DELETE (queue) :: = Item

توجه داریم که Queue مجموعهٔ تمام صفهایی است که در این فروشگاه وجود دارد و queue یک صف مخصوص مثلاً صف لوازم آرایشی است.

همین طور Item مجموعهٔ تمام مشتریان صف است و item یک فرد مخصوص مثلاً آقای احمدی است.

قوانین

قوانینی که بر ساختمان دادهٔ صف حکفرماست عبارتند از:

۱. از صف خالی نمی توان فردی حذف کرد.
۲. به صفی که پر شده نمی توان فرد اضافه کرد.

ادامه دارد

زمان لازم برای اجرای الگوریتم ترانهاده سریع برابر $O(n+t)$ می‌باشد زیرا

$O(n + t + n + t)$ → برای استفاده از a و b

n برای تعداد دهی آغازین Row size

t برای تشکیل Row size

n برای Rowstart

(۱)

C_A:	6	7	8
1	5	3	
2	1	-2	
4	1	4	
4	5	-1	
5	2	3	
5	5	2	
5	6	7	
6	4	2	

Row size

1	2	3	4	5	6	7
~~0~~ 1 2	~~0~~ 1	0	~~0~~ 1	~~0~~ ~~1~~ 2 3	~~0~~ 1	0

→

2	1	0	1	3	1	0

(۲)

Rowstart

1	2	3	4	5	6	7
1	3	4	4	5	8	9

Row start

1	2	3	4	5	6	7
~~1~~ ~~2~~ 3	~~3~~ 4	4	~~4~~ 5	~~5~~ ~~6~~ ~~7~~ 8	~~8~~ 9	9

Row start(1) := 1;

Row start[i] := Row start[i - 1] + Row size (i - 1)

(۳)

C_{AT}:	7	6	8
1	1	2	-2
2	1	4	4
3	2	5	3
4	4	6	2
5	5	1	3
6	5	4	-1
7	5	5	2
8	6	5	7

$A =$

0	0	4	0	0	0	0	0	0
-5	0	0	0	0	4	0	0	0
0	0	0	0	6	-1	0	0	0
0	0	4	0	0	0	0	0	0
0	3	1	0	0	0	0	2	0
0	0	0	1	0	0	0	0	5

6x9

نمایش آرایه‌ها (تهیه‌کننده: مشوه)

همانطور که می‌دانید آرایه تعدادی خانه هم جنس و متوالی در حافظه است که همگی یک اسم داشته باشند و در زبانهای سطح بالا نظیر پاسکال یا C برای رسیدن به یک خانه مخصوص در آرایه کافی است از اندیس آن خانه استفاده کنیم، اما در زبانهای سطح پایین نظیر اسمبلی از آرایه فقط آدرس اولین خانه را داریم و برای آنکه به یک خانه مخصوص دسترسی پیدا کنیم باید حتماً در خانه‌ای حافظه با استفاده از آدرس خانه اول پیش برویم تا به خانه مورد نظر برسیم. یک آرایه n بعدی در زبان پاسکال در حالت کلی به شکل

$$a[l_1 \ldots n_1, l_2 \ldots n_2, \ldots, l_n \ldots n_n]$$

می باشد، تعداد عناصر چنین آرایه‌ای برابر است با:

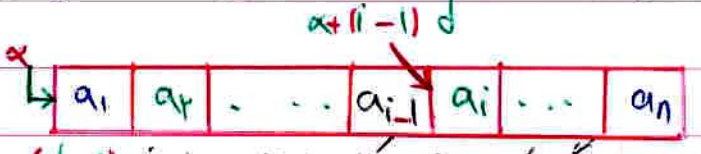
$$\prod_{i=1}^{n} (n_i - l_i + 1)$$

در حالت خاص اگر محدوده پایین تمام ابعاد را (l_i) برابر یک در نظر بگیریم ($l_i = 1$) در این صورت تعداد عناصر آرایه حاصلضرب n_n تا n_1 خواهد بود در ادامه سعی می کنیم آدرس یک خانه مخصوص از آرایه را در مورد آرایه‌ای یک بعدی و دو بعدی و سه بعدی تعیین کنیم.

الف) آرایه‌های یک بعدی

شکل کلی این آرایه‌ها به صورت $a[l \ldots u]$ می باشد، اگر آدرس اولین خانه آرایه برابر α باشد، برای پیدا کردن آدرس خانه i ام باید $(i-1)$ خانه پیش برویم و اگر فرض کنیم هر خانه آرایه به طول d بایت باشد آنگاه آدرس a_i به صورت زیر به دست می آید:

$$\text{آدرس } a_i = \alpha + (i-1)\,d$$



برای سادگی کار فرض می کنیم l نیز ۱ باشد $(l=1)$

ب) آرایه‌های دو بعدی

شکل کلی این آرایه‌ها به صورت $a[1 \ldots n_1, 1 \ldots n_2]$ می باشد، برای پیدا کردن آدرس خانه سطر i ام و ستون j ام همان (a_{ij}) ابتدا باید از عناصر $i-1$ سطر عبور کنیم، از آنجاکه در هر سطر n_2 عنصر داریم، آدرس ابتدای سطر i ام به صورت زیر تعیین می شود.

$$\text{آدرس ابتدای سطر } i = \alpha + (i-1)\,n_2$$

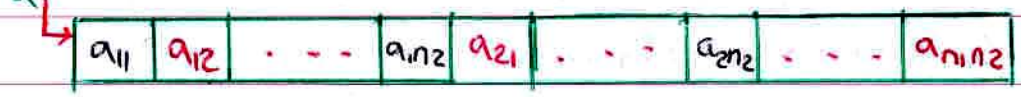
$$a = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n_2} \\ a_{21} & a_{22} & \cdots & a_{2n_2} \\ \vdots & & & \\ a_{n_1 1} & a_{n_1 2} & \cdots & a_{n_1 n_2} \end{bmatrix}$$

در سطر i ام برای رسیدن به عنصر j ام مشابه آرایه‌های یک بعدی باید از $(j-1)$ عنصر عبور کنیم، پس آدرس a_{ij} به صورت زیر به دست می‌آید:

$$\text{آدرس } a_{ij} = \alpha + (i-1)n_2 + (j-1)$$

تذکر:

عناصر آرایه دوبعدی a به صورت سطری در حافظه ذخیره می‌شود یعنی ابتدا عناصر سطر اول ذخیره می‌شود سپس عناصر سطر دوم و الی آخر یعنی در حافظه ماتریس یا آرایه دوبعدی a به شکل زیر ذخیره می‌گردد:

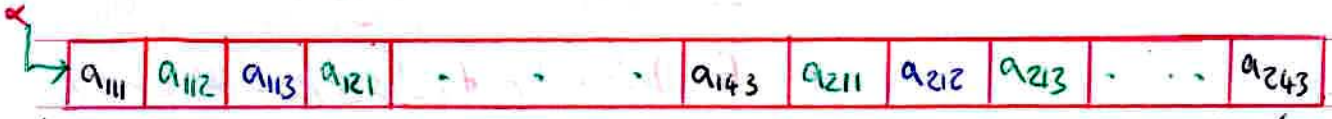


ج) آرایه‌ی سه بعدی

شکل کلی این آرایه‌ها به صورت $a[l_1..u_1, l_2..u_2, l_3..u_3]$ می‌باشد مثلاً اگر $a[1..2, 1..4, 3..5]$ یک آرایه سه بعدی باشد آنرا به صورت زیر در نظر می‌گیریم

(i: 1..2، j: 1..4، k: 3..5)

$$i=1 \Rightarrow \begin{bmatrix} a_{111} & a_{112} & a_{113} \\ a_{121} & a_{122} & a_{123} \\ a_{131} & a_{132} & a_{133} \\ a_{141} & a_{142} & a_{143} \end{bmatrix} \qquad i=2 \Rightarrow \begin{bmatrix} a_{211} & a_{212} & a_{213} \\ a_{221} & a_{222} & a_{223} \\ a_{231} & a_{232} & a_{233} \\ a_{241} & a_{242} & a_{243} \end{bmatrix}$$



در این گونه آرایه‌ها ابتدا عناصر صفحه (ماتریس اول) ذخیره می‌شود سپس عناصر صفحه دوم و همین طور در نهایت عناصر صفحه n_1 ام برای آنکه آدرس a_{ijk} را به دست آوریم ابتدا باید از عناصر $i-1$ صفحه یا ماتریس عبور کنیم و چون در هر صفحه به اندازه n_2n_3 عنصر وجود دارد آدرس ابتدای صفحه i ام برابر است با:

$$\text{آدرس ابتدای صفحه } i \text{ ام} = \alpha + (i-1)\, n_2n_3$$

از اینجا به بعد مانند آرایه‌ای دوبعدی عمل می‌کنیم در صفحه i ام برای رسیدن به سطر j ام باید از $j-1$ سطر عبور کنیم و هر سطر n_3 عنصر دارد.

$$\text{آدرس ابتدای سطر } j \text{ ام در صفحه (ماتریس) } i \text{ ام} = \alpha + (i-1)\, n_2n_3 + (j-1)\, n_3$$

و در نهایت آدرس a_{ijk} برابر است با:

$$\alpha + (i-1)\, n_2n_3 + (j-1)\, n_3 + (k-1)$$

بسمه تعالی

صف (QUEUE) و پشته (STACK)

به طور کلی برای بازیابی داده‌های ترتیبی از دو شیوه می توانیم استفاده نمیم:

در شیوهٔ اول

اولین دادهٔ ورودی اولین داده ای است که بازیابی می شود (FIFO=First In First Out)
این شیوهٔ بازیابی را صف (QUEUE) می نامیم. در کامپیوتر نیز از این ساختار به وفور استفاده می شود
مثلاً هنگام ترجمهٔ یک برنامه توسط مترجم ابتدا خط اول ترجمه می شود سپس خط دوم الی آخر یا در حافظه‌ی میانگیر (BUFFER) نیز از سیستم صف استفاده می شود.

در شیوهٔ دوم

بازیابی اطلاعات ترتیبی استفاده از پشته (STACK) می باشد، در پشته آخرین داده ورودی اولین داده خروجی است (LIFO=Last In First Of) این ساختار نیز کاربردهای فراوانی در کامپیوتر دارد مثلاً در مورد پرانتزهای تودرتو اولین پرانتزی که بسته می شود مربوط به آخرین پرانتز است که باز شده است یا حلقه‌های تودرتو آخرین حلقه‌ای که باز می شود یا شروع می شود اولین حلقه‌ای است که به اتمام می رسد.

ساختمان داده‌های پشته و صف در صفحات ۹۸ و ۱۰۱ آمده است:

صفحه ۱۰۱ ساختار ساختمان داده صف

Objects عناصر: تعدادی متناهی لیست مرتب با صفر عنصر یا بیشتر

Functions توابع

Positive integer اعداد صحیح مثبت

Creat (نام صف و اندازه صف) ← یک صف خالی به نام queue ایجاد می کند که حداکثر اندازهٔ آن ماکزیمم سایز است.

Is Full تابع خروجی True و False است ← اگر تعداد افراد عناصر در صف مساوی ماکزیمم سایز شده درست در غیر این صورت غلط است.

Add تابع

* else عنصر item را در انتهای صف queue درج می کند و صف جدید در اختیار قرار می دهد.

قانون: هر صف باید انتها ← آن مشخص باشد و انتها یک متغیر برای rear
← هر موقع نفر است اضافه شود به ته صف اضافه می شود.

تابع Is Empty صفی خالی است که تعداد عناصرش صفر باشد و در غیر این صورت پُر است و (MaxSize = 0) است.

تابع Delet اسم صفی می‌خواهد و خروجی از جنس Item و چک می‌شود که صف خالی نباشد.

قانون: عنصر ابتدای صف را از صف خارج می‌کند و آن را ارجاع می‌دهد. (Return = ارجاع)

جلو Front

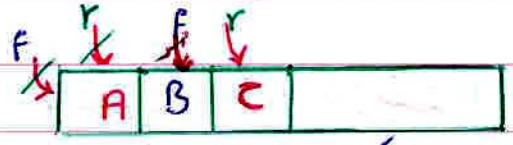
هیچ وقت از نفر سوم صف کسی خارج نمی‌شود.

صفحه ۱۰۲

تابع Create در صحت var یک آرایه است و Front و Rear صفر می‌شوند.

Rear همیشه به آخرین نفر اشاره می‌کند.

Front به یکی قبل از اولین نفر



در این صف یک نفر آدم C

تذکرات مربوط به ساختمان داده صف

۱. هر صف نیاز به دو متغیر (پارامتر) دارد، یکی Front برای تعیین مکان شروع صف (ابتدای صف)، دیگری Rear که انتهای صف را مشخص می‌کند.

۲. همواره Rear به آخرین نفر صف اشاره می‌کند، اما Front یکی قبل از اولین نفر صف را مشخص می‌کند.

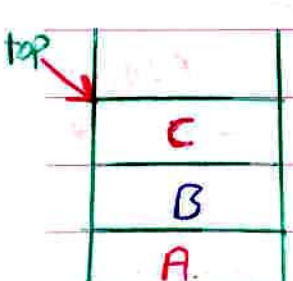
۳. هنگامی صف خالی است که Rear مساوی Front باشد.

۴. هنگامی صف پُر است که Rear مساوی n یا Max Size باشد.

ساختار پشته

نکات مربوط به ساختمان داده پشته

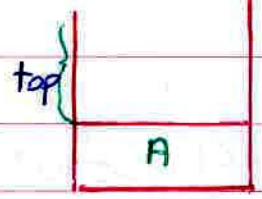
۱. اولین نکته: در ساختار پشته فقط یک پارامتر داریم و آن top است که همواره به عنصر بالای پشته اشاره می‌کند.



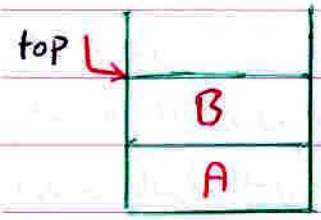
۲. هنگامی پشته خالی است که پشته مساوی صفر باشد. (top = 0)

۳. هنگامی پشته پُر است که top مساوی MaxSize باشد. (top = MaxSize)

۴. تفاوت عمده ساختار صف با پشته:




```
ADD:
top:=top+1;
Stack[top]:=item;
```




```
DELET:
item:=Stack[top];
top:=top-1;
```

تواجع مربوط به اضافه کردن و کم کردن از صف و پشته در صفحات ۱۰۲، ۹۹، ۱۰۰ آمده است.

عمل اضافه کردن عنصر به پشته Push و کم کردن عنصر POP است.

ارزشیابی عبارات

عبارانی نظیر 7 * 6 + 5 از نظر محاسباتی عبارات مبهم هستند، زیرا اگر ابتدا 5 و 6 را جمع کرده سپس حاصل را در 7 ضرب کنیم نتیجه 77 خواهد بود، اما اگر ابتدا عمل ضرب را انجام دهیم سپس عمل جمع حاصل 47 می شود.

یک راه ساده برای رفع ابهام این گونه عبارات پرانتزگذاری است، در پرانتزگذاری می توان اولویت عملگرها را تعیین کرد اما این کار در ریاضیات نوسوم نیست، در ریاضیات عمل ضرب نسبت به جمع مقدم است بنابراین عبارت a+bc هیچ گونه ابهامی ندارد، در کامپیوتر یا ماشین حساب نیز می توانیم برای عملگرهای مختلف اولویت تعیین کنیم مثلاً آنجا نیز بگوئیم ضرب نسبت به جمع مقدم است.

اما کامپیوتر یا ماشین حساب چگونه می تواند اولویت عملگرها را اعمال کند؟

برای جواب دادن به این سؤال باید با ساختار عبارات بیشتر آشنا شویم. اعمال ریاضی معمولاً اعمال دوتایی هستند یعنی هر عملگر روی دو عملوند (OPERAND) اثر می کند مثلاً برای جمع دو مقدار A و B می نویسیم A+B در این شکل نوشتار که عملگر در میان دو عملوند آمده است شکل میانوند (INFIX) داریم، اگر عملگر را پیش از دو عملوند بنویسیم چنین +AB شکل پیشوند (PREFIX) خواهیم داشت، و اگر عملگر را پس از دو عملوند بنویسیم شکل پسوند (POSTFIX) داریم در این حالت می نویسیم AB+، در عمل برای رفع ابهام عبارات ریاضی در کامپیوتر یا ماشین حساب آنها را به صورت پسوندی تبدیل می کنیم سپس حاصل عبارات را با استفاده از یک پشته محاسبه می کنیم.

برای آنکه یک عبارت را از حالت میانوند به حالت پسوند تبدیل کنیم دو راه وجود دارد:
یکی به صورت مستقیم و دیگری با استفاده از پشته

در روش مستقیم با توجه به جدول اولویت عملگرها که در صفحه 114 آمده است ابتدا ترتیب انجام اعمال را مشخص می کنیم، سپس هر عمل را به صورت POSTFIX تبدیل می کنیم و در نهایت پرانتزها را حذف می کنیم.

تمام عباراتی که به پسوند تبدیل شده اند شکل پسوندی کل عبارت را تعیین می کنیم.

در جدول اولویت عملگرها

در سیستم توان هم، (not) است.

U = UNARY = تفریق یکانی (منفی) U

5-3 دوتایی

U 7

برای برعکس کردن بیتهای ۰ و ۱ از Xor استفاده می‌نیم.

Xor	۰	۱
۰	۰	۱
۱	۱	۰

یای قانع جمع

```
  11010001   xor
  11111111
  --------
  00101110   ← مکمل
```

* in در مجموعه‌ای زبان پاسکال همان عضویت است. $in = \in$

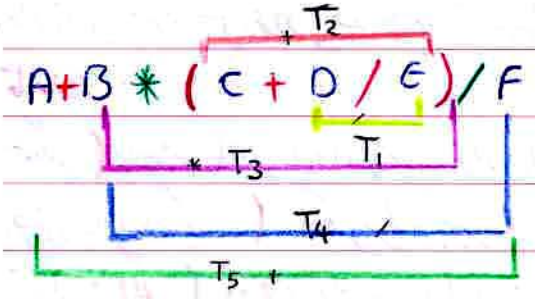
نکته جدول صفحه ۱۱۴:

در بین عملگرهای هم‌اولویت آن عملگری که سمت چپ باشد مقدم است.

اولویت	عملگر
1	( ) پرانتز
2	U تفریق یکنایی (منفی)
3	not
4	*, /, div, mod, and
5	+, −, or, xor
6	in, >, >=, =, <>, <=, <

اولویت عملگرها در پاسکال

مثال: عبارت زیر را به صورت پسوندی بنویسید.



T_1: D/E ⟶ DE/

T_2: C+T_1 ⟶ C T_1 + ≡ CDE/+

T_3: B * T_2 ⟶ B T_2 * ≡ BCDE/+*

T_4: T_3 / F ⟶ T_3 F/ ≡ BCDE/+*F/

T_5: A + T_4 ⟶ A T_4 + ≡ ABCDE/+*F/+ شکل POSTFIX عبارت بالا

روش قبل بسیار ساده است اما دو ایراد اصلی دارد:

یکی آنکه عبارت باید دوبار پیمایش شود بار اول عملگرهای موجود در آن تعیین می شوند و اولویت آنها مشخص می شود بار دوم عبارت تبدیل به پسوندی می شود. این عملیات ضمن آنکه زمان گیر می باشد گاهی اوقات حتی انجام پذیر نیست مثلاً در ماشین حساب کل عبارت در دست نیست تا تمام عملگرهای موجود در آن اولویت بندی شوند.

برای رفع این مشکل:

برای رفع این مشکل از روشی استفاده می کنیم که عملاً در ماشین حساب و کامپیوتر انجام می شود.

در این روش از یک پشته (STACK) برای تبدیل عبارات میانوندی به پسوندی استفاده می کنیم.

به دو نکته اساسی توجه داریم:

اول اینکه در عبارت پسوندی هیچ پرانتزی نداریم.

دوم آنکه ترتیب عملوندها از شکل میانوندی به پسوندی تغییر نمی کند یعنی تنها جای عملگرها عوض می شود.

قوانین زیر را رعایت می کنیم:

قانون ۱. تنها عملگرها داخل پشته قرار می گیرند و هیچ عملوندی را در پشته نمی نویسیم به محض رسیدن به یک عملوند آن را مستقیماً در خروجی می نویسیم.

قانون ۲. تنها عملگری می تواند در پشته بالای عملگر دیگر قرار بگیرد که اولویتی بیش از آن داشته باشد اگر عملگر جدید اولویتی کمتر یا مساوی عملگر بالای پشته داشته باشد ابتدا آن عملگر از پشته خارج می شود سپس عملگر جدید در پشته قرار می گیرد.

قانون ۳.

عملگر) پرانتز باز بالاترین اولویت را داراست این عملگر می تواند در پشته بالای هر عملگر دیگری (حتی بالای یک) دیگر قرار بگیرد.

تذکر: عملگر) دو نوع اولویت دارد:

یکی اولویت جاری (ICP) و دیگری اولویت در پشته (ISP).

اولویت جاری عملگر) از همه بالاتر است اما به محض اینکه در پشته قرار گرفت پایین ترین اولویت را خواهد داشت یعنی اگر عملگر) داخل پشته بود هر عملگر دیگری می تواند بالای آن قرار بگیرد.

قانون ۴. برحسب رسیدن به عملگر (کلیه عملگرهای موجود در پشته را از پشته خارج می‌نمیم تا به اولین عملگر پرانتز باز برسیم آن عملگرها را در خروجی می‌نویسیم) را نیز از پشته خارج می‌نمیم ولی در خروجی نمی‌نویسیم.

قانون ۵. هنگام رسیدن به انتهای عبارت که با نماد # (number sine) نشان داده می‌شود پشته را خالی می‌کنیم یعنی تمام عملگرهای موجود در آن را خارج می‌سازیم و در خروجی می‌نویسیم.

مثال: عبارت قبل را با استفاده از پشته به POSTFIX تبدیل کنید.

POST I

A	+	B	*	(	C	+	D	/	E	)	/	F	#

tokenize جزء بندی

عبارت جزء next token	پشته stack	خروجی out put
بعدی	#	بعدی
A	#	A
+	# +	A
B	# +	AB
*	# + *	AB
(	# + * (	AB
C	# + * (	ABC
+	# + * (+	ABC
D	# + * (+	ABCD
/	# + * (+ /	ABCD
E	# + * (+ /	ABCDE
)	# + *	ABCDE/+
/	# + /	ABCDE/+*
F	# + /	ABCDE/+*F
#	خالی	ABCDE/+*F/+

۳. همواره اولین تقاضا از صف خارج می شود.

۴. همواره فرد جدید به انتهای صف اضافه می شود.

تذکر: ممکن است تواج دیگری نیز بر این ساختمان داده اضافه کنیم مثلاً تاج merge برای ادغام دو صف در یکدیگر یا تاج split برای شکاف یک صف به دو صف کوتاهتر و برای این تواج نیز می توانیم قانون تعیین کنیم.

«بسمه تعالی» «۸۱/۷/۱۵»

مقایسه الگوریتم ها

الگوریتم های مختلف را از دو جهت می توان مقایسه کرد:

یکی حافظه مصرفی و دیگری سرعت اجرای الگوریتم ها. حافظه مورد نیاز هر الگوریتم را به دقت می توان تعیین کرد مثلاً می دانیم هر متغیر صحیح ۲ بایت و هر متغیر اعشاری ۴ بایت و هر متغیر کاراکتری یا حرفی ۱ بایت حافظه می گیرد. ضمن اینکه حافظه مورد نیاز برای دستورالعمل ها را نیز می دانیم بدین ترتیب می توان بدقت تعیین کرد که یک الگوریتم چند بایت حافظه مصرف می کند. بین دو الگوریتمی که کار یکسانی انجام می دهند می توانیم آن را که حافظه کمتری مصرف می کند تعیین کنیم. اما در این درس به حافظه مصرفی الگوریتم ها توجه چندانی نداریم زیرا حافظه کامپیوترهای کنونی بسیار زیاد است ضمن اینکه افزایش حافظه بسادگی امکان پذیر است (حافظه ارزان می باشد)

اما نکته مهمتر هنگام مقایسه الگوریتم ها بررسی سرعت آنهاست، بدیهی است هر چه تعداد دستورالعمل ها در یک الگوریتم کمتر باشد آن الگوریتم سریعتر است. بنابراین ابزاری نیاز داریم تا شمار یا تعداد دستورالعمل ها را تعیین کند این ابزار تاج O نام دارد.

تعریف دقیق و ریاضی تاج O به صورت زیر است:

اگر دنباله $|\alpha_n|$ و اگر a باشد (به سمت بی نهایت برود) آنگاه می گوئیم مرتبه بی نهایت آن با تندی مرتبه دنباله $|n^p|$ است، اگر عدد مثبتی مانند K وجود داشته باشد به طوری که

$$\exists K>0 \quad \lim_{n\to\infty} \frac{|\alpha_n|}{n^p} \leq K$$

نماد آن که برای مرتبه به کار می رود: $\alpha_n = O(n^p)$

زیر برنامه POSTFIX در صفحه 119 آمده است:

e: expression ورودی e که یک عبارت است.

توضیح: خروجی شکل پسوندی عبارت میانوندی e می باشد. NextToken عرضه شابه برنامه eval می باشند. فرض شده که آخرین جزء عبارت e، # باشد ضمناً # در ته پشته قرار داده شده است.

Stack[1]:='#'; top:=1; مقداردهی آغازین پشته

در حلقه while دوم (داخلی): isp در پشته را با icp مقایسه می کند و icp جاری وارد پشته می خواهد شود.

تا زمانی که عملگر جدید، یکی یکی از بالای پشته Delete می کند و در خروجی چاپ می کند.

صفحه 116 eval

PROC مقداردهی کردن

یک عبارتی تبدیل به POSTFIX شده چه طوری توان ارزشیابی کرد؟

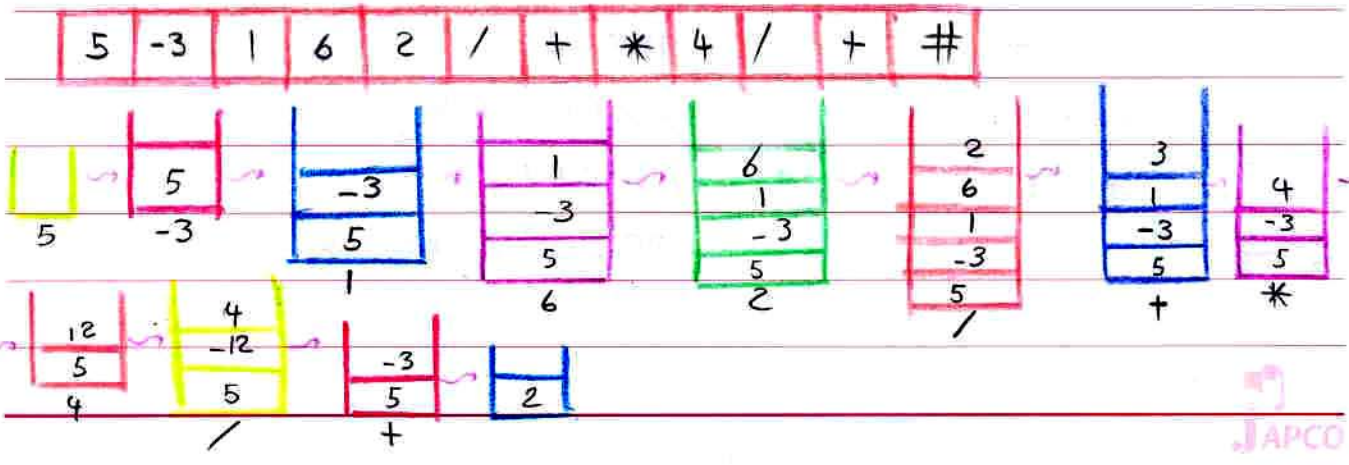
تذکر: با استفاده از پشته می توان عبارتی را که به صورت پسوندی تبدیل شده ارزشیابی نمود، عملوندها را در پشته قرار می دهیم و به محض رسیدن به یک عملگر از بالای پشته به اندازه مورد نیاز عملوند خارج می کنیم و پس از آنکه آن عمل روی عملوندها انجام شد حاصل را با لای پشته می نویسیم، این کار را ادامه می دهیم تا به انتهای عبارت پسوندی برسیم.

مثال:

عبارت مثال قبل را به ازای مقادیر زیر بدست آورید.

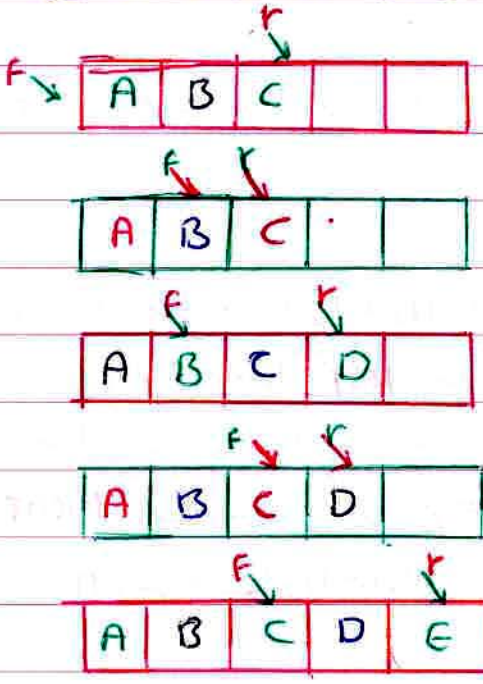
AT4 + ≡ ABCDE / + * F / +

A=5, B=-3, C=1, D=6, E=2, F=4



صف حلقوی

در صفهای عادی ممکن است در اثر صف پر نباشد اما با پیغام پر بودن صف مواجه شویم که یک پیغام کاذب است مثلاً صفی به طول ۵ را در نظر بگیرید که بخواهیم عملیات زیر را روی آن انجام دهیم:



۱. A و B و C وارد می شوند.

۲. A و B خارج می شوند.

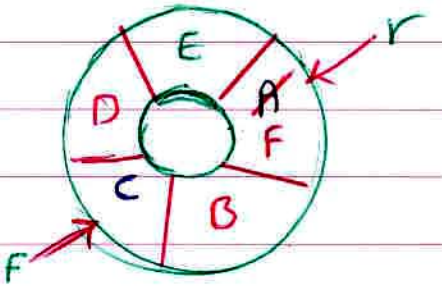
۳. D وارد می شود.

۴. C خارج می شود.

۵. E و F وارد می شوند.

برای ورود F با پیغام پر بودن صف مواجه می شویم.

در واقع در این صف تنها دو نفر داخل صف هستند (D و E) اما چون که Rear مساوی Max Size شده است پیغام پر بودن صف صادر می شود. اولین راهی که برای رفع این مشکل به ذهن می رسد آن است که فرد جدید یعنی F را به جای A قرار دهیم یعنی به صورت منطقی فکر کنیم خانه جدا از خانه پنجم، خانه اول است بعبارت دیگر صف را به صورت حلقوی یا دایره ای در نظر بگیریم.



در ریاضیات برای آنکه بگوییم عددی مقدار بیش از n نداشته باشد یعنی جدا از n به یک بر n می توانیم از Mod استفاده کنیم یعنی رابطه زیر را قرار دهیم :

محدوده از ۱ تا n

```
rear := (rear mod n) + 1
```

زیر برنامه مربوط به صف حلقوی در صفحه ۱۰۵ آمده است و محدوده تغییرات از ۰ تا n-1 است .

تذکر مهم :

در صف های حلقوی یک مشکل بوجود می آید و آن این است که پر و خالی بودن صف قابل تشخیص نیست ، در این صف اگر front مساوی rear باشد ممکن است صف کاملاً پر یا کاملاً خالی باشد در برای رفع این مشکل یک راه آن است که :

بین rear و front یک خانه خالی بگذاریم یعنی اجازه ندهیم rear به front برسد در این صورت به جای n خانه از صف از (n-1) خانه استفاده می کنیم

(طول صف در مثال قبل ۵ است ولی ۴ تا استفاده می کنیم .)

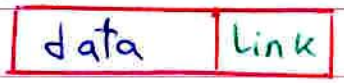
لیست های پیوندی

در فصل های قبل با ساختار آرایه آشنا شدیم. آرایه تعدادی خانه هم‌نام متوالی و هم جنس در حافظه می‌باشند، هم ترتیب و ضعیت آنها بین ترتیب مشخص آنهاست ولی گاهی اوقات این قرینه به ضعف تبدیل می‌شود مثلاً فرض کنید در آرایه‌ای به طول ۱۰۰۰ بخواهیم عنصر دهم را حذف کنیم برای این کار باید عنصر یازدهم را به جای دهم بنویسیم سپس دوازدهم را به جای یازدهم بنویسیم و همین طور عنصر هزارم را به جای ۹۹۹ بنویسیم، مثلاً باید ۹۹۰ خانه آرایه هرکدام یکی به سمت چپ منتقل شود (شیفت به چپ) در همین آرایه اگر بخواهیم بین خانه پنجم و ششم یک خانه جدید اضافه کنیم مثلاً باید ۹۹۵ خانه به سمت راست منتقل شوند تا جا برای خانه جدید ایجاد شود. مثلاً در ساختارهایی که عملیات حذف و اضافه در آنها زیاد باشد استفاده از آرایه مقرون به صرفه نیست، بهتر است که:

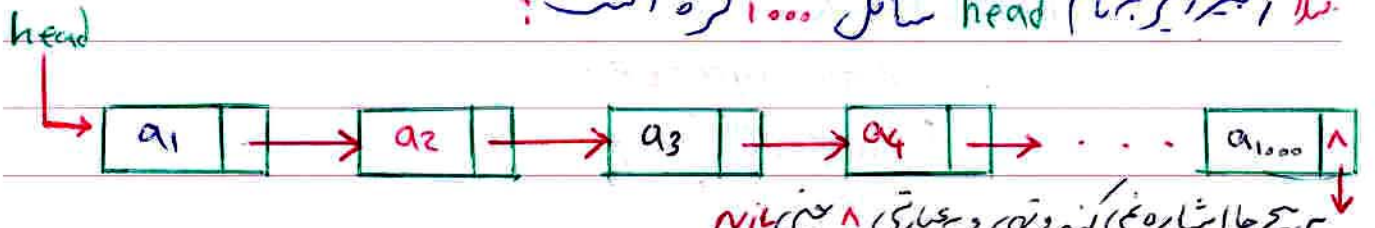
ترتیب خانه‌ها را به شکل منطقی در نظر بگیریم یعنی خانه‌های اول تا هزارم به صورت فیزیکی پشت سر هم نباشند چنین ساختاری را لیست پیوندی (LINKED LIST) می‌نامیم.

هر لیست پیوندی یا زنجیر (CHAIN) از تعدادی گره (NODE) تشکیل شده که در آن هر گره حداقل دو قسمت دارد:

یکی قسمت data و دیگری ربط یا پیوند (LINK) که به گره بعدی اشاره می‌کند.



مثلاً زنجیر زیر به نام head شامل ۱۰۰۰ گره است:



به هیچ جا اشاره نمی‌کند و تهی و بعبارتی ^ یعنی Nil

ساختار node

در زبان پاسکال می‌توانیم ساختار node را به صورت زیر معرفی کنیم. این ساختار در قسمت نوع داده (TYPE) با استفاده از دستور رکورد به صورت زیر معرفی می‌شود:

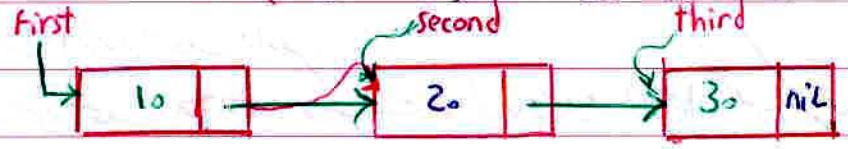
صفحه بعد

ساختار node

```
Pointer = ^ node ;
node = RECORD
        data : INTEGER ;
        Link : Pointer ;
       END ;
```

Pointer در واقع اشاره گری به یک ساختار از نوع node می باشد. بنابراین باید ساختار اشاره گر را نیز در قسمت TYPE همقبل از معرفی ساختار node تعریف نمیم. علامت اشاره گر در زبان پاسکال ^ (hat) می باشد.

مثال: برنامه ای که یک لیست پیوندی با سه گره به نامهای First, second, third به شکل زیر ایجاد می گردد. (تذکر: در برنامه کلمات کلیدی حروف بزرگ و متغیرها حروف کوچک)




```
PROGRAM  linktest ;
TYPE     pointer = ^node ;
           node = RECORD
                   data : INTEGER ;
                   link : pointer ;
                  END ;
VAR
   first, second, third : pointer ;
BEGIN
   NEW (first) ;          → یک محفظه برای first ایجاد می کند.
   first^.data := 10 ;
   NEW (second) ;
```

ادامه

```
First ^. link := second ;
Second ^. data := 20 ;
NEW (third) ;
Second ^ .link := third ;
third ^ .data := 30 ;
third ^ .link := NIL ;
END .
```

تذکر:
برنامه مثال قبل هیچ خروجی روی صفحه نمایش ندارد اما زنجیر را در حافظه ایجاد می‌کند.

تذکر:
عملاً مهم‌ترین کاربرد لیست‌های پیوندی پویایی آنهاست یعنی اینکه بتوانیم برزمان که به یک گره نیاز داشتیم آنرا ایجاد کنیم. در برنامه زیر این پویایی به کار گرفته می‌شود. در برنامه زیر تعدادی عدد حقیقی نامنفی از ورودی خوانده می‌شود و توسط آنها یک لیست پیوندی ایجاد می‌شود که اولین گره آن با اشاره گر head مشخص شده است. سپس زنجیر پیمایش شده و مجموع مقادیری در گره‌های آن وجود دارد تعیین می‌شود فرض می‌کنیم اولین عدد حتماً نامنفی باشد.

در این برنامه
Pointer را با اختصار ptr و data را d و link را L قرار می‌دهیم.

```
PROGRAM no2;
TYPE   Ptr = ^ node ;
       node = RECORD
                d : REAL;
                L : PTR ;
              END;
VAR
   head, P, Q : Ptr ;
   x , Sum : REAL;
```

```
BEGIN
  READLN (X);
   NEW(head);
  head^.d := X;
   P := head;
  READLN (X);
WHILE   (X >= 0)   DO
  BEGIN
   NEW (q);
    q^.d := X;
    P^.L := q;
      P := q;
   READLN (X);
END;
   P^.L := NIL;

  P := head ;
Sum := 0 ;
WHILE   ( P <> NIL)  DO
   BEGIN
    Sum := Sum + P^.d ;
      P := P^.L;
   END;
WRITELN ( ' Sum = ' , Sum : 10 : 2) ;
END.
```

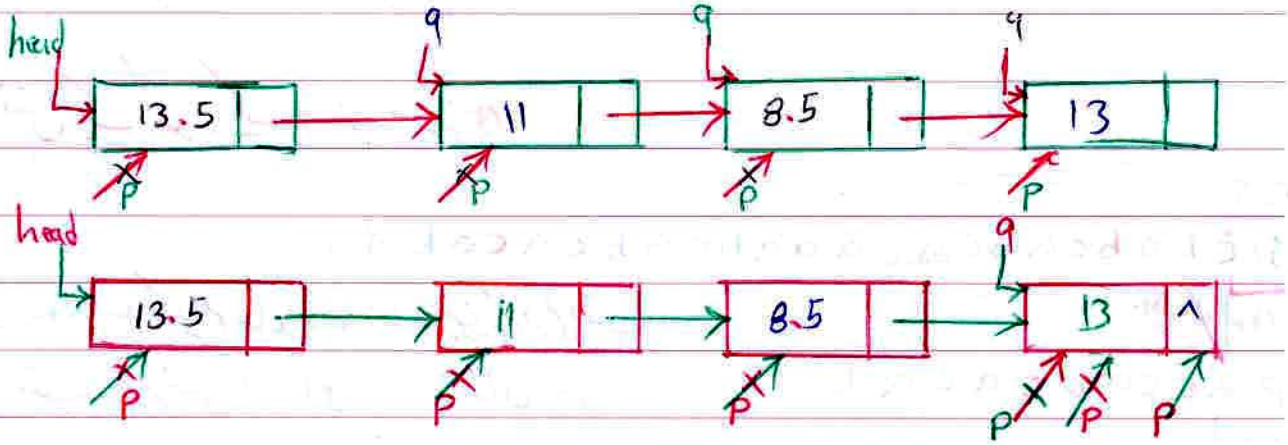
لیست ایجاد می شود

لیست پیمایش می شود و جمع تقاضیر در Sum قرار می گیرد.

ادامه

مثال قبل را به ازای $x = 13.5, 11, 8.5, 13, -1$ ردیابی (TRACE) نمایید.

x	شرط حلقه WHILE	Sum	خروجی
~~13.5~~		∅	Sum = 46
~~11~~	TRUE	13.5	
~~8.5~~	TRUE	24.5	
~~13~~	TRUE	33	
-1	FALSE	46	



تمرین :

برنامه‌ای بنویسید که یک لیست پیوندی از تعدادی عدد حقیقی نامنفی ایجاد کند سپس لیست را پیمایش کرده و بزرگترین عدد موجود در لیست را به همراه کوچکترین عدد لیست و همچنین میانگین اعداد داخل لیست در خروجی چاپ کند.

«بسمه تعالی»

رشته

صفحه ۸۳

concat الحاق دو رشته

substr(S, i, j): string

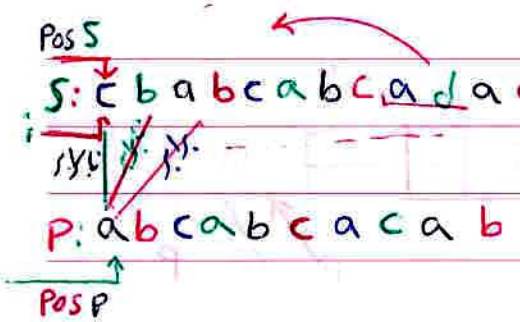
S: a b c d e f g h

substr(S, 1, 3); → a b c

از کاراکتر i ام رشته S به طول j جدا می کند مثلا در این مثال اگر بگوییم:

substr(S,4,2) ← d, e می دهد.

یافتن یک الگو در یک رشته صفحه ۸۴



یک abc پیدا کردیم به اندازه abc جلو می گردیم عقب،

این عقب و جلو رفتن ها در این رشته زمان گیر

زمان اجرا O(lengh P . lengh S)

فلوچارت P برگشت به عقب و در S به این صورت نیست و برگشت به عقب ندارد و جلو می رود.

مثال: در رشته مقابل

A B C D E F G H I J

اینجا برای پیدا کردن GH مشکلی نیست چون یکی بیشتر نیست.

تاج شکست

برنامه تاج شکست در صفحه ۸۷

تاج شکست در واقع یک بردار یکسان است.

اگر در کاراکتر اول شکست بخوریم باید به خانه اول برگردیم بنابراین خانه اول صفر است.

ادامه

باتوجه برنامه صفحه ۸۷

درکار استر ۵ شکست خورد به کار استر ۲ بیایید.

j:	1	2	3	4	5	6	7	8	9	10
f:	∅	∅	∅	1	2	3	4	∅	1	2
P:	a	b	c	a	b	c	a	c		

از اینجا بررسی کردیم ابتدا

باتوجه به P و S

i	j
~~0~~	~~2~~
~~1~~	~~3~~
~~2~~	~~4~~
~~3~~	~~5~~
~~4~~	~~6~~
~~1~~	~~7~~
~~6~~	~~8~~
	~~9~~
	~~10~~

شکست Failure

خط ۶: محاسبه می کند تابع شکست را برای الگوی P

اگر در کار استر چهارم شکست خوردی بیا به کار استر اول چون یکسانند.

صفحه ۸۶

match مطابق بودن (یکسان بودن)

زیر رشته Pat در رشته S و تابع f

(معرف match maker)

	1	2	3	4	5	6	7	8	9	10
P:	a	b	c	a	b	c	a	c	a	b

POS S →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
S:	c	b	a	b	c	a	b	c	a	b	a	a	a	b	c	a	b	c	a	c	a	b

حلقه را تا زمانی ادامه می دهیم که یکی از POS P یا POS S تمام شود.

POS S	POS P
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	5
10	6
11	3
12	1
13	2

(اساری نیست سراغ کار استر سوم اول)

(اوقتی اساری نیست برگرد یکی جلو کار دوم)

(اساری نیست)

روی کار استر پنجم می آییم

POS P := f(POS P − 1) + 1 = 5

خط 13

مثال: مرتبه دنباله $\alpha_n = \left| \frac{4n^4 - 5}{n+3} \right|$ را تعیین کنید.

$$\lim_{n\to\infty} \frac{|\alpha_n|}{n^p} = \lim_{n\to\infty} \frac{\left|\frac{4n^4-5}{n+3}\right|}{n^p} = \lim_{n\to\infty} \frac{|4n^4-5|}{n^{p+1}+3n^p}$$

$$p+1 = 3 \Rightarrow p = 2$$

$$\Rightarrow \lim_{n\to\infty} \frac{|\alpha_n|}{n^p} = \lim_{n\to\infty} \frac{|4n^4-5|}{n^4+3n^2} = 1 \quad k = 4 \leftarrow$$

$$\hookrightarrow \frac{4n^4-5}{n+3} = O(n^3) \quad , \quad k = 4$$

تست کنکور

مثال: زمان اجرا یا شمار فقطه برنامه زیر را تعیین کنید.

```
x := 0;
FOR i:=1 To n DO
  FOR j:=1 To n DO
    BEGIN
      x := x + 1;
      y := z * x;
      z := y - x;
    END;
  WRITELN(x , y , z);
```

$$1 + \left(\sum_{i=1}^{n} \sum_{j=1}^{n} 3 \right) + 1 = 2 + 3 \sum_{i=1}^{n} \sum_{j=1}^{n} 1 = 2 + 3 \sum_{i=1}^{n} n = 2 + 3n \sum_{i=1}^{n} 1 = 2 + 3n^2$$

پس $O(n^2)$ با ثابت $k = 3$ است.

در پایان برنامه مقدار x برابر n^2 است.

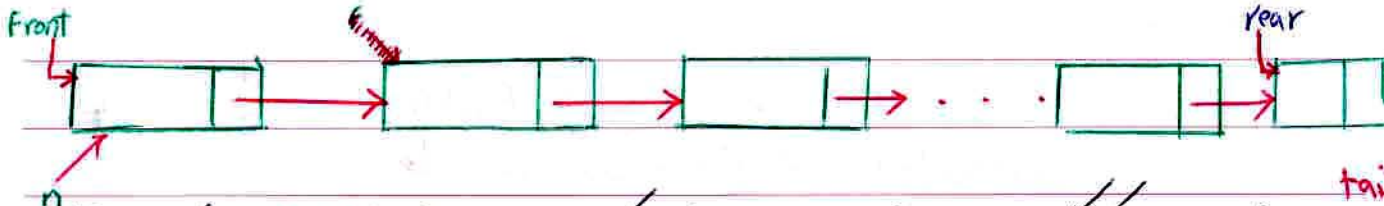
مثال: زمان اجرا برنامه زیر را تعیین کنید.

```
FOR i := 1 To n DO
  FOR j := 1 To i DO
    x := x + 1;
```

جواب ←

صف و پشته پیوندی

در فصل قبل دیدیم که با استفاده از آرایه چگونه می‌توان ساختار صف را شبیه‌سازی کرد با استفاده از لیست پیوندی نیز چنین کاری امکان‌پذیر است. در یک صف پیوندی دو اشاره‌گر لازم داریم که یکی به ابتدای صف اشاره کند (Front) و دیگری به انتهای صف (rear یا tail)



tail

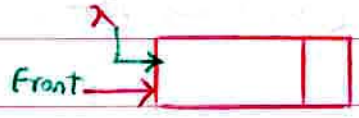
به راحتی می‌توان یک گره جدید به انتهای صف اضافه کرد زیر برنامه صفحه ۱۳۷ این کار را انجام می‌دهد (زیر برنامه Add Queve)

یک گره جدید به نام x می‌گیرد و دو پارامتر i و y به پروسیجر ارسال می‌شود.

توضیح {{

گره y را به صف i ام اضافه می‌کند.

شرط برای این است که اگر صف خالی بود، گره x جدید Front و غیر این صورت اگر صف خالی نبود باید rear شود.

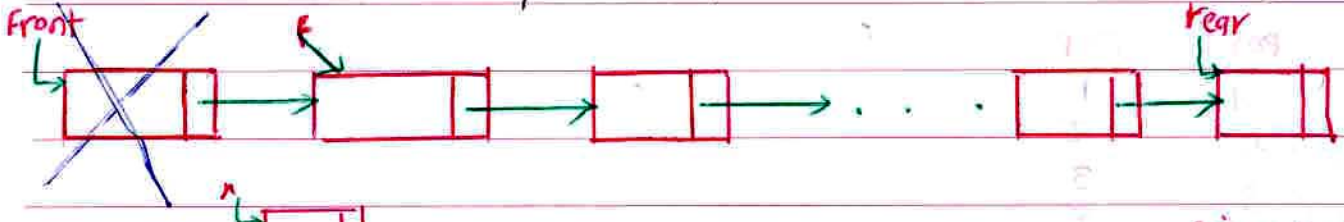


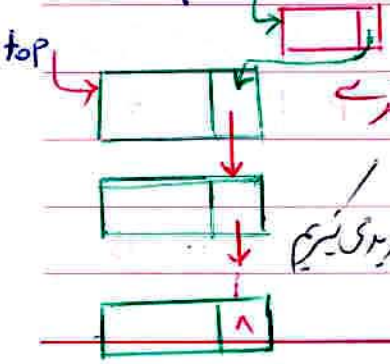
Delet

dispose حافظه را آزاد می‌کند.

زیر برنامه حذف یک عنصر از یک صف نیز در صفحه ۱۳۸ آمده است:

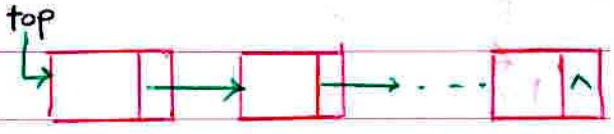
وقتی یک تعداد صف برمی‌داریم یک Data را حذف می‌کنیم.





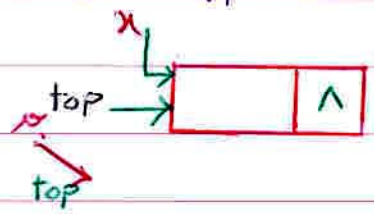
در مورد پشته

یا



در مورد پشته پیوندی نیز عملیات به سادگی امکان پذیر است ، در صفحه ۱۳۷ زیر برنامه‌های Add Stack و Delet Stack عملیات اضافه و حذف را انجام می‌دهد.

* چون هنگام ایجاد پشته top ← nil است بنابراین چک می‌کنیم پشته خالی است یا نه.



چند جمله‌ایهای پیوندی

در فصلهای قبل با استفاده از آرایه‌ها توانستیم چند جمله‌ایها را ذخیره کنیم و عملیات ساده ریاضی را روی آنها انجام دهیم. با استفاده از لیست‌های پیوندی نیز به راحتی می‌توان چند جمله‌ایها را شبیه سازی کرد برای این منظور هر جمله یک چند جمله‌ای را یک گره از لیست پیوندی می‌گیریم که سه قسمت دارد یکی قسمت ضریب که عدد حقیقی است دیگری قسمت توان که عدد صحیح است و سومی اشاره‌گر به گره بعدی (جمله بعدی)

گره چند جمله‌ای Polynode

coef	exp	link

زیر برنامه PAdd در صفحه ۱۴۲ جمع دو چند جمله‌ای را انجام می‌دهد.

attack = ملحق کردن ضمیمه می‌کند

attach کار همان Newterm را انجام می‌دهد.

«بسمه تعالی»

«سوالات امتحان میان ترم»

۱.

```
FOR  i:= 1  To  n  DO
BEGIN
    j:=2*i ;    k:= n-i;
    WHILE   j <=k      DO
            j:=۲ *j ;
END ;
```

2
4
8
16
32
64
128
256
512
1024

(n-۲i)

$n \log_2^n =$

۴.

```
F(Q):= IF   ISEMPTY(Q)     THEN Q;
F(ADDQ(i,Q)) := IF  ISEMPTY(Q)   THEN   ADDQ(i,Q);
                ELSE  ADDQ(DELETQ, F(DELETQ));
```

$Q = \{\} \Rightarrow F(Q) = \{\}$

$Q = \{A\} = ADDQ('A', \{\}) \Rightarrow F(ADDQ('A', \{\})) = ADDQ('A', \{\}) = \{A\}$

$Q = \{A,B\} = ADDQ('B', \{A\}) \Rightarrow F(ADDQ('B', \{A\})) = ADDQ('A', F(\{\})) = ADDQ('A', \{\}) = \{A\}$

$Q = \{A,B,C\} = ADDQ('C', \{A,B\}) \Rightarrow F(ADDQ('C', \{A,B\})) = ADDQ('A', F\{B\}) = ADDQ('A', \{B\})$

$= \{B, A\}$

گرافها و درختها

هر گراف G از دو مجموعه تشکیل شده است یکی V و دیگری E:

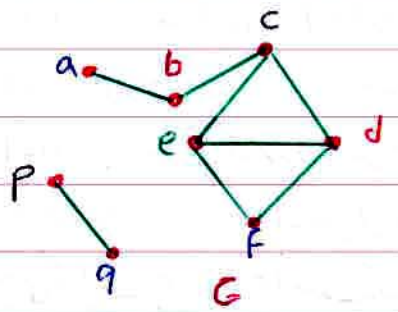
V یک مجموعه غیرتهی و متناهی است که آنرا مجموعه رئوس یا گره‌ها یا نقاط گراف می‌نامیم و E نیز تعدادی از زیر مجموعه‌های دو عضوی V است، E را مجموعه یالها یا لبه‌ها یا خطوط گراف می‌نامیم.

در هر گراف اگر بین دو رأس یال وجود داشته باشد آنها را مجاور می‌نامیم و تعداد رئوسی را که با یک رأس دلخواه مجاورند درجه آن رأس می‌نامیم. مثلاً در گراف زیر رأس b درجه ۲ است و رأس d درجه ۳ می‌باشد.

یالها

$G = (V, E)$

رئوس



$V = \{a, b, c, d, e, f, P, 9\}$

اگر بتوانیم از یک رأس گراف شروع کرده و با گذشتن از یالها و رئوس گراف به یک رأس دیگر برسیم به شرط آنکه، هیچ رأس و یالی تکراری نباشد می‌گوییم بین دو رأس مسیر وجود دارد.

اگر بین هر دو رأس دلخواه گراف حداقل یک مسیر وجود داشته باشد آن گراف را همبند یا مرتبط می‌نامیم

مثال قبل مثالی از یک گراف همبند (مرتبط) بود زیرا بین P و e، هیچ مسیری وجود ندارد.

اگر از یک رأس شروع کرده و با گذشتن از یالها و رئوس بتوانیم مجدداً به همان رأس بازگردیم بدون آنکه رأس و یال تکراری وجود داشته باشد، آنگاه یک دوره خواهیم داشت.

مثلاً در گراف قبل $c \longrightarrow d \longrightarrow e \longrightarrow c$ یک دوره به طول ۳ است زیرا سه یال دارد.

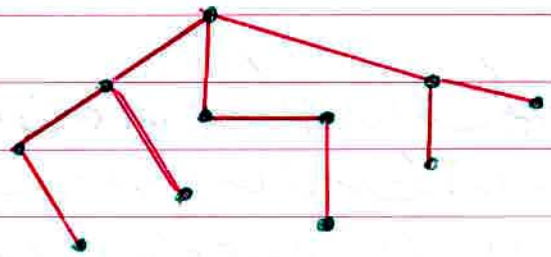
$d \longrightarrow f \longrightarrow e \longrightarrow c \longrightarrow d$ یک دوره به طول ۴ است.

تعریف درخت

هر گراف همبند و بدون دوره را درخت می‌نامیم.

مثال ←

مثال :



درخت مقابل ۱۱ رأس و ۱۰ یال دارد.

- نقطه یک درخت یک رأس دارد و بدون یال

نکات زیر در مورد درختها هم ارزی (معادلند) :

۱. G یک درخت است.
۲. بین هر دو رأس دلخواه G فقط یک مسیر منحصر بفرد داریم.
۳. G یک گراف همبند n رأسی است که n-۱ یال دارد.
۴. G یک گراف n رأسی با n-۱ یال است که همبند باشد.
۵. G یک گراف n رأسی با n-۱ یال است که دوره نداشته باشد.
۶. G یک گراف n رأسی بدون دوره است که n-۱ یال دارد.
۷. G می نیمال همبند است یعنی گراف همبند است که اگر یک یال آن برداشته شود ناهمبند می شود.
۸. G یک گراف ماکزیمال نادوره است یعنی با اضافه شدن یک یال به آن حتماً دوره خواهد داشت.

درخت ریشه دار را به صورت بازگشتی تعریف کنید به صورت زیر :

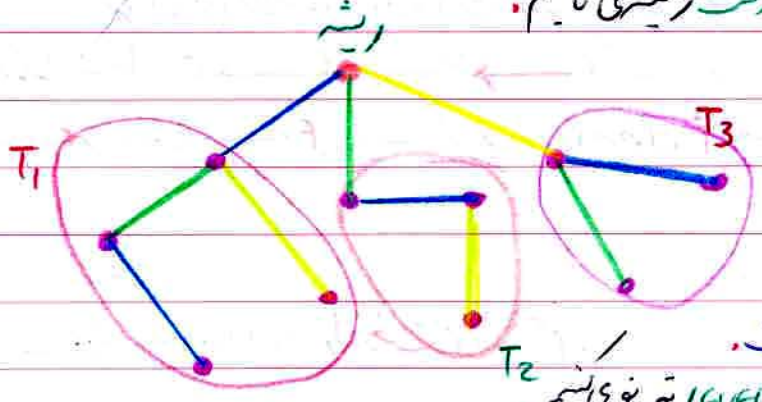
تعریف :

درخت مجموعه‌ای از چند گره به صورت زیر می باشد :

الف) گره خاصی به نام ریشه دارد.

ب) بقیه گره ها به جز ریشه به مجموعه‌های $T_1, T_2, \ldots, T_n$ افراز شده اند به طوری که هر کدام از T_i ها یک درخت هستند و هر T_i را یک زیردرخت ریشه‌دار نامیم.

مثال :



* در درخت هر گره را می توان ریشه گرفت.

در هر درختی که ریشه داشته باشد سطح یا LEVEL تعریف می کنیم.

تعاریف مقدماتی در درخت

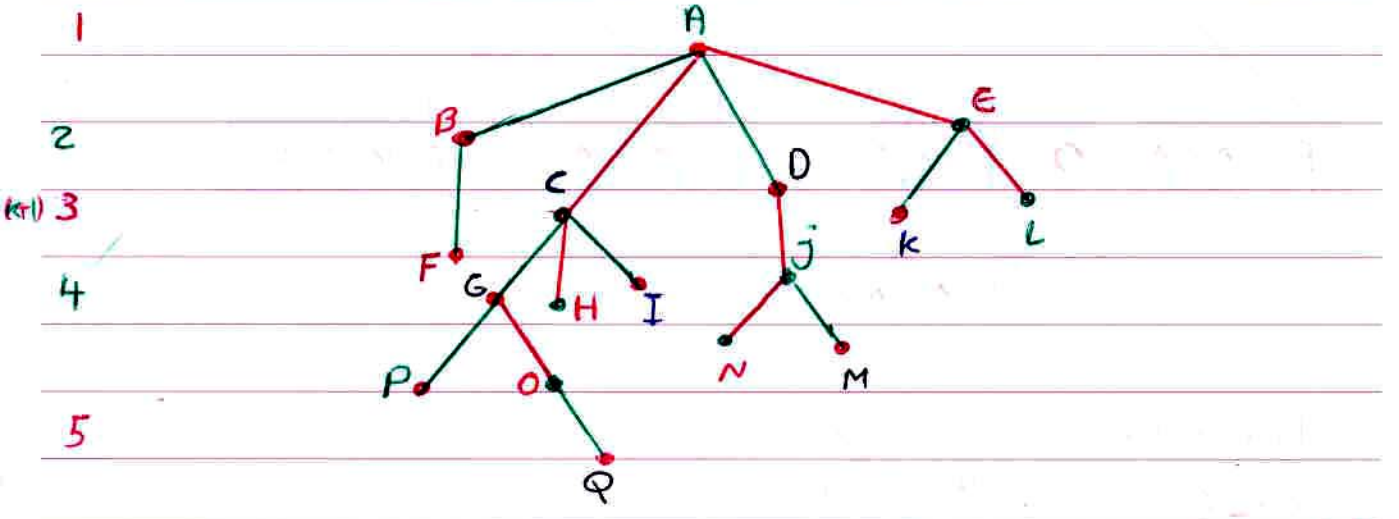
در هر درخت ریشه را در سطح (LEVEL) یک تعریف می کنیم و مقام رئوسی را که فاصله آنها تا ریشه برابر k باشد در سطح k+1 در نظر می گیریم، اگر یک رأس مخصوص در سطح k ام با رئوسی در سطح k+1 ام مجاور باشد رأس سطح k را پدر یا والد رئوس سطح k+1 می نامیم و آن رئوس سطح k+1 را فرزندان رأس سطح k ام نامگذاری می کنیم، خود فرزندان هم نیا، نامیده می شوند مثلاً در درخت زیر رأس C سه فرزند با نامهای G و H و I دارد، بنابراین G و H و I هم نیا هستند توجه داریم که I و J هم نیا نیستند.

حداکثر سطح در یک درخت را عمق درخت می نامیم مثلاً درخت زیر عمق ۵ است.

در هر درخت درجه رأس برابر تعداد فرزندان وی است مثلاً رأس C درجه ۳ است و رأس D درجه یک می باشد.

رأس k نیز درجه صفر است زیرا فرزندی ندارد، رئوس درجه صفر را برگ می نامیم.

* تعریف درجه در گراف و درخت با هم متفاوت است.



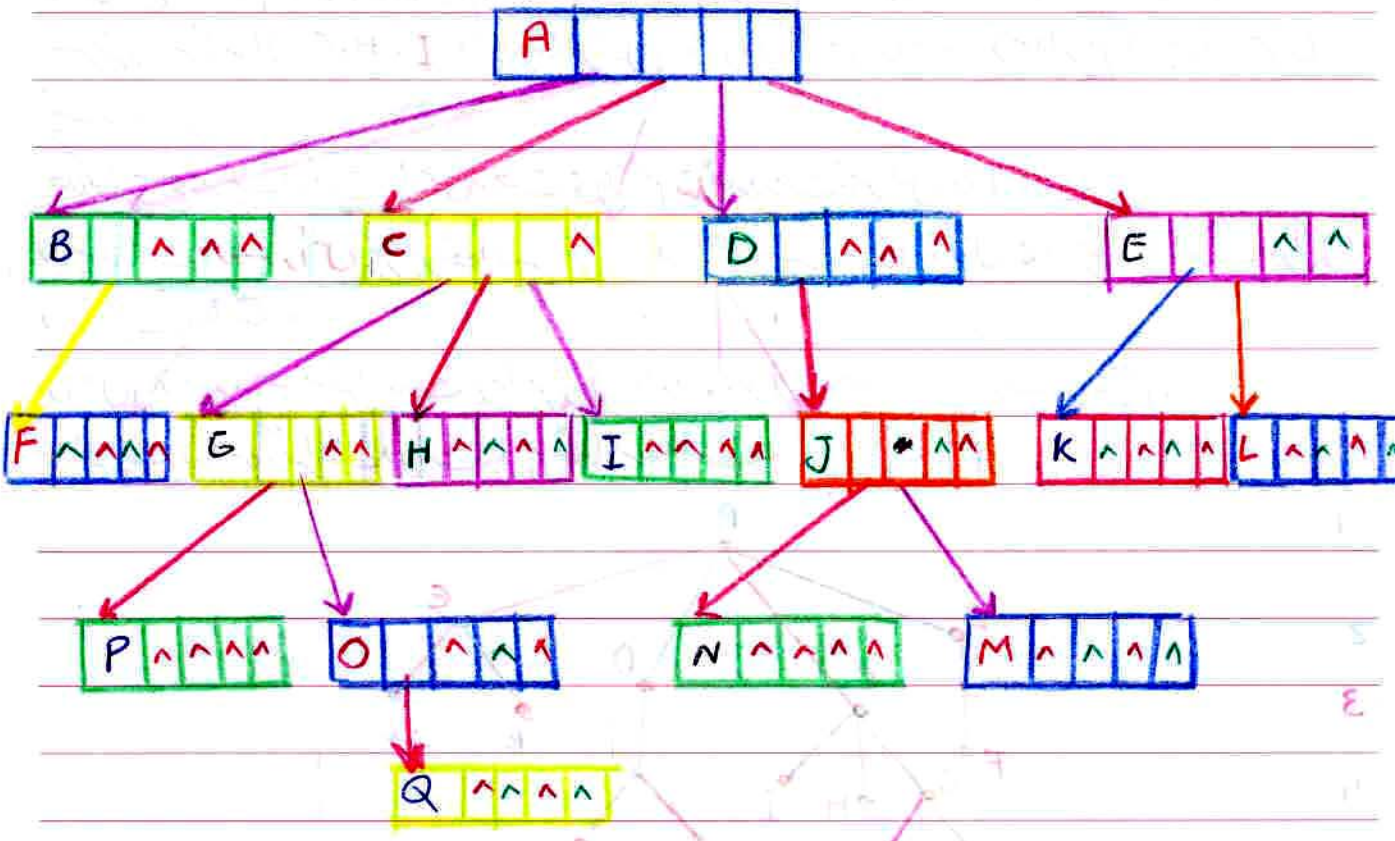
نمایش درخت

دو ابزار اساسی برای نمایش درخت وجود دارد یکی آرایه و دیگری لیست های پیوندی، معمولاً برای نمایش درخت از آرایه استفاده نمی شود.

برای استفاده از لیست پیوندی باید گره ای تعریف کنیم که یک قسمت Data و چند قسمت پیوند (link) داشته باشد، تعداد link ها در واقع برابر با بیشترین درجه رئوس درخت است، در مثال قبل بیشترین فرزندان را A دارد که ۴ فرزند است پس باید گره ای ۵ قسمتی تعریف کنیم که یک قسمت Data دارد و ۴ اشاره گر به نامهای ch1 تا ch4 که به فرزندان گره اشاره می کنیم.

صفحه بعد ←

با این حساب درخت مثال قبل را می‌توان به شکل زیر نشان داد:



$17 \times 4 = 68$

$68 - 16 = 52$

تعداد اشاره‌گر با NIL

تعداد اشاره‌گرهای استفاده نشده (تعداد یال‌ها)

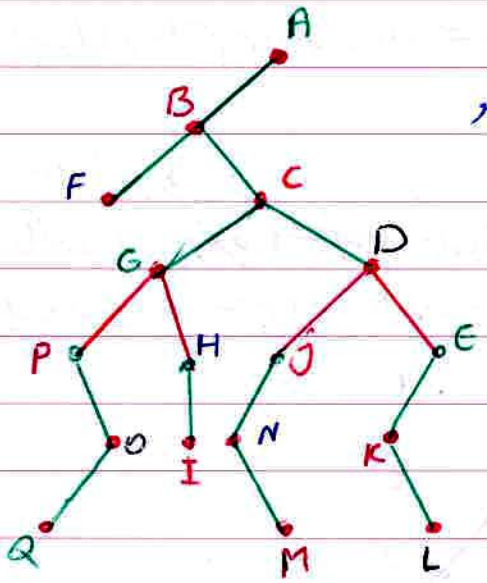
تمرین

در یک درخت با n رأس و فاکتور هم درجه d چند اشاره‌گر NIL وجود دارد؟

همانطور که مشاهده می‌شود تعداد اشاره‌گرهای NIL در روش قبل بسیار زیاد است به همین خاطر از روش دیگری استفاده می‌کنیم که در آن تعداد اشاره‌گرهای NIL کاهش یابد. این روش جدید روش فرزند چپ - هم نیای راست نامیده می‌شود.

در این روش هر گره حداکثر با دو گره در سطح بعد مجاور است که این گره‌ها یکی فرزند چپ گره است و دیگری هم نیای راست گره است.

گره	فرزند چپ	هم نیای راست



عمق درخت افزایش یافته و برابر ۷ است.
تعداد اشاره‌گرهای NIL کاهش یافته.

اشاره‌گرهای NIL $17 \times 2 - 16 = 18$

اگر n را رأس داشته باشیم در روش فرزند چپ - هم نیای راست کلاً $n+1$ اشاره‌گر پوچ خواهیم داشت.

تمرین

چگونه می‌توان از گرافی که به صورت فرزند چپ - هم نیای راست نمایش داده شده است به گراف اصلی رسید؟

درختهای دودویی (BINARY TREE)

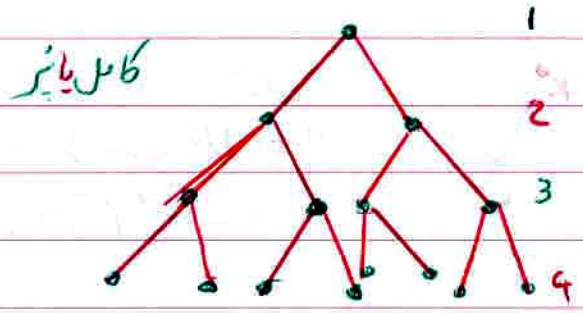
درروش فرزند چپ ـ هم نیا راست به درختی رسیدیم که در آن هر گره حداکثر ۲ فرزند دارد، چنین درختهایی درخت دودویی نام دارند، این درختها در علوم مختلف بسیار پرکاربردند در واقع هر جا از منطق و ارزشی استفاده کنیم می‌توان درخت دودویی را به کار برد، در کامپیوتر نیز چون کار مبنای محاسبات مبنای ۲ است این درخت‌ها کاربرد دارند.

نمایش درختهای دودویی

از آنجایی که درختهای دودویی نسبت به درخت‌های عادی منظم‌تر هستند، می‌توانیم آنها را در آرایه نیز نمایش دهیم، ابتدا باید بدانیم هر درخت دودویی به چند خانه آرایه نیاز دارد.

تعریف درخت دودویی پر یا کامل

درخت دودویی را که در آن هر گره ۰ یا ۲ (به جز برگها) فرزند داشته باشد درخت دودویی پر می‌نامیم.
مثلاً درخت زیر یک درخت دودویی پر با عمق ۴ است.



درختهای پر دارای خواص زیر هستند:

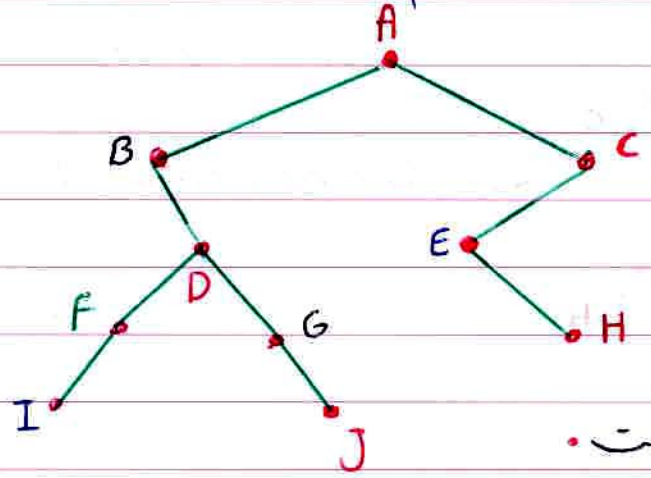
خاصیت اول) تعداد گره‌ها در سطح k ام برابر 2^{k-1} است.

۲. اگر عمق درخت d باشد تعداد کل گره‌های درخت دودویی پر $2^d - 1$ است.

$$2^0 + 2^1 + 2^2 + \cdots + 2^{d-1} = 2^d - 1$$

با استفاده از این خواص برای نمایش یک درخت دودویی به عمق d از آرایه‌ای به طول $2^d - 1$ استفاده می‌کنیم، ریشه را در خانه اول آرایه قرار می‌دهیم و به صورت بازگشتی فرزند چپ خانه i ام را در خانه (2i) قرار می‌دهیم و فرزند راست خانه i ام را در خانه 2i+1 قرار می‌دهیم.

مثال: درخت دودویی زیر را در یک آرایه نمایش می‌دهیم.



$d=5 \Rightarrow 2^d - 1 = 31$

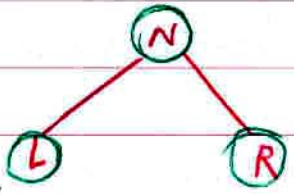
آرایه ← به طول 31 نیاز است.

نکته: پدر گره خانه i ام در خانه $\lfloor \frac{i}{2} \rfloor$ است.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	B	C		D	E				F	G		H							I			J								

پیمایش در خت‌های دودویی

درخت‌های دودویی را به ۲ صورت عمده می‌توان پیمایش کرد یکی پیمایش سطحی و دیگری پیمایش عمقی، در این قسمت پیمایش‌های عمقی را در نظر می‌گیریم، شکل کلی هر درخت دودویی به صورت زیر می‌باشد.



این سه گره را به $3!$ (فاکتوریل) حالت پیمایش (باز یابی) کرد، ما قرارداد می‌کنیم که فرزند چپ همواره مقدم بر فرزند راست باشد.

در این صورت ۳ حالت پیمایش خواهیم داشت:

در حالت اول ابتدا گره بازیابی می‌شود سپس فرزندان چپ و راست (NLR) این حالت پیمایش را حالت پیش ترتیبی (PRE ORDER) می‌نامیم.

اگر گره n در میان فرزندان چپ و راست پیمایش شود (LNR) آنگاه پیمایش میان ترتیبی (INORDER) خواهیم داشت و در صورتی که گره n پس از فرزندانش بازیابی شود (LRN) پیمایش پس ترتیبی (POSTORDER) خواهیم داشت. در این حالت هیچ گره‌ای بازیابی نمی‌شود مگر آنکه تمام فرزندانش بازیابی شده باشند به همین جهت ریشه آخرین گره‌ای است که بازیابی می‌شود.

در بعضی کتابها به جای N از D استفاده می‌شود.

$$\sum_{i=1}^{n}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = 1+2+\dots+n = \frac{n(n+1)}{2} = \frac{n^2}{2}+\frac{n}{2}$$

پس $O(n^2)$ با $k=\frac{1}{2}$

فرمولهای مهم

روابط زیر در مورد پیدا کردن زمان اجرا بسیار مفید می باشد:

$$\sum_{i=1}^{n} 1 = n \longrightarrow O(n^1),\ k=1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \longrightarrow O(n^2),\ k=\frac{1}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \longrightarrow O(n^3),\ k=\frac{1}{3}$$

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} \longrightarrow O(n^4),\ k=\frac{1}{4}$$

$$\vdots$$

$$\sum_{i=1}^{n} i^m \longrightarrow O(n^{m+1}),\ k=\frac{1}{m+1}$$

مثال) سوال امتحان

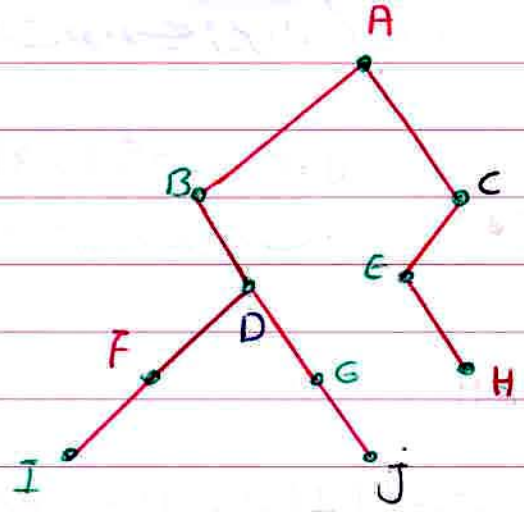
```
FOR i:=1 To n DO
 FOR j:=1 To i DO
  FOR k:=1 To (i*j) DO
   x:=x+1;
```

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{ij} 1 = \sum_{i=1}^{n}\sum_{j=1}^{i} ij = \sum_{i=1}^{n} i \times \sum_{j=1}^{i} j = \sum_{i=1}^{n} i \times \frac{i(i+1)}{2} = \frac{1}{2}\sum_{i=1}^{n}(i^3+i^2)$$

$$= \frac{1}{2}\left[\frac{n^2(n^2+1)}{4} + \frac{n(n+1)(2n+1)}{6}\right] \Rightarrow$$ پس $O(n^4)$ با ثابت $k=\frac{1}{8}$

مثال

پیمایش درخت زیر را بنویسید.



ابتدا فرزندان چپ بازیابی بعد فرزندان راست

پیمایش‌های عمقی

NLR: A, B, D, F, I, G, J, C, E, H

LNR: B, I, F, D, G, J, A, E, H, C

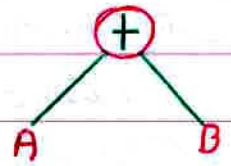
LRN: I, F, J, G, D, B, H, E, C, A

کاربرد درخت‌های دودویی

همانطور که گفته شد درخت‌های دودویی بسیار پرکاربردند، یکی از کاربردهای آنها در تبدیل عبارات ریاضی است، اغلب اعمال ریاضی، اعمال دوتایی هستند یعنی یک عملگر روی دو عملوند اثر می‌کند، یکی عملوند چپ و دیگری عملوند راست، بنابراین متناظر هر عبارت ریاضی می‌توانیم یک درخت دودویی داشته باشیم.

مثلاً درخت متناظر با عبارت A+B به صورت زیر نمایش داده می‌شود:

(در عملگر خطا می‌کشیم)



عملگر را در ریشه قرار می‌دهیم و عملوندهای چپ و راست را جای فرزندان قرار می‌دهیم. عبارات پیچیده‌تر را نیز می‌توان متناظر با یک درخت دودویی گرفت، با این نکته اساسی توجه داریم که هر عملی که اولویت بالاتری دارد سطح بالاتری دارد یعنی از ریشه دورتر است و عملگر با کمترین اولویت در ریشه قرار می‌گیرد.

ارائه صفحه جدید

مثال

درخت متناظر با عبارت زیر را رسم کنید.

$A+B*(C+D/E)/F$

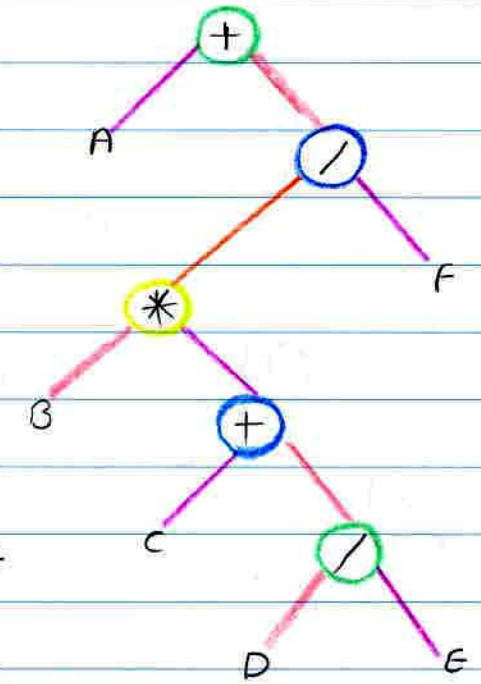
T_1: D/E

T_2: $C+D/E$

T_3: $B*(C+D/E)$

T_4: $A+B*(C+D/E)$

T_5: $A+B*(C+D/E)/F$



(PREORDER) NLR: + A / * B + C / D E F

LNR:

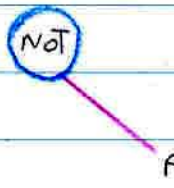
LRN: A D E / C + B * F / +

صفحات ۲۳۰ و ۲۳۲ زیر برنامه‌های inorder, Pre order, Postordere جهت پیمایش های میان ترتیبی و پیش ترتیبی و پس ترتیبی آمده است.

صفحه ۲۳۰ inorder بازگشتی و صفحه ۲۳۲ بدون استفاده از بازگشتی.

اعمال یکتایی (-A) و (NOT A) است و در اعمال یکتایی فرزندش، فرزند راست است.

-A → A- ; NOTA

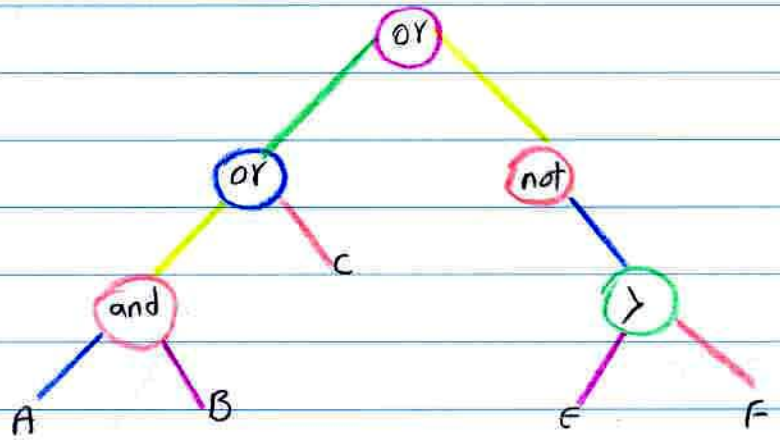


(در امتحان POSTFIX پایشه و INFIX با درخت)

صفحه ۱۱۸ درخت رسم و Postorder پیمایش شود.

ث) A and B or C or not (E>F)

T_3: A and B
T_1: E>F
T_2: not (E>F)
T_4: A and B or C or not (E>F)



(postorder): LRN: AB and C or EF > not or

(Preorder): NLR: or or and ABC not > EF → عبارت Prefix

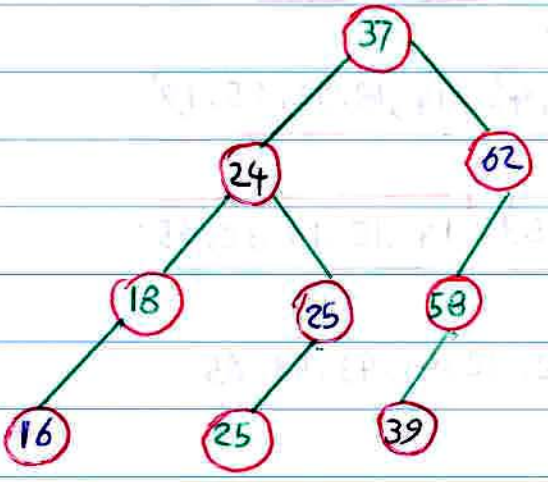
"بسمه تعالی"

کاربرد درختهای دودویی در مرتب سازی

یکی از روشهای مرتب سازی روش مرتب سازی کومه‌ای (HEAP SORT) می‌باشد. این روش جزء روشهای پویا (Dynamic) است یعنی حتی اگر تعداد داده‌ای که می‌خواهیم مرتب کنیم از ابتدا مشخص نباشد باز هم می‌توانیم از این روش استفاده کنیم، در این روش اولین داده را در ریشه قرار می‌دهیم سپس به صورت بازگشتی در مورد داده‌های دیگر اگر از مقدار گره جاری کوچکتر یا مساوی بودند به سمت چپ گره می‌رویم و اگر مقدار داده جدید بزرگتر از گره جاری بود به سمت راست آن می‌رویم، در نهایت یک درخت دودویی ایجاد می‌شود که با پیمایش INORDER آن داده‌ها به صورت صعودی مرتب می‌شوند.

مثال: داده‌های زیر را به روش مرتب سازی کومه‌ای مرتب کنید.

37, 24, 62, 18, 25, 16, 58, 39, 25



INORDER (LNR): 16, 18, 24, 25, 25, 37, 39, 58, 62

زیربرنامه HEAP SORT در صفحه ۳۷۶ آمده است:

زمان اجرای مرتب سازی کومه‌ای $O(n \log_2^n)$ می‌باشد.

نکته: زمان اجرای مرتب سازی حبابی $O(\frac{1}{2}n^2)$

اگر: مرتب سازی حبابی $n = 2^{20}$ ⟶ زمان اجرا $= 500 \times 10^9$

اگر: مرتب سازی کومه‌ای $n = 2^{20}$ ⟶ زمان اجرا $= O(2^{20} \times 20)$

مرتب سازی ادغامی (MERGE SORT)

اساس کار این روش ادغام آرایه‌های مرتب در یکدیگر است، بدین صورت که در **تکرار اول** هر دو عدد کنار هم را به ترتیب صعودی مرتب می‌کنیم تا در واقع آرایه‌های مرتب به طول ۲ بدست آید، در **تکرار دوم** آرایه‌های مرتب به طول ۲ را که کنار هم هستند در هم ادغام می‌کنیم تا آرایه‌های مرتب به طول ۴ بدست آید، در **تکرار سوم** آرایه‌های چهارتایی را ادغام می‌کنیم تا مرتب ۸ تایی بدست آید، این کار را ادامه می‌دهیم تا کل آرایه مرتب شود.

مثال:

41, 29, 53, 16, 19, 32, 17, 43, 65, 14, 18, 32, 19

41, 29 | 53, 16 | 19, 32 | 17, 43 | 65, 14 | 18, 32 | 19

29, 41 | 16, 53 | 19, 32 | 17, 43 | 14, 65 | 18, 32 | 19

16, 29, 41, 53 | 17, 19, 32, 43 | 14, 18, 32, 65 | 19

16, 17, 19, 29, 32, 41, 43, 53 | 14, 18, 19, 32, 65

14, 16, 17, 18, 19, 19, 29, 32, 32, 41, 43, 53, 65

به عنوان تمرین ثابت کنید

زمان اجرای مرتب سازی ادغامی $O(n \log_2^n)$ است.

مزیت این روش

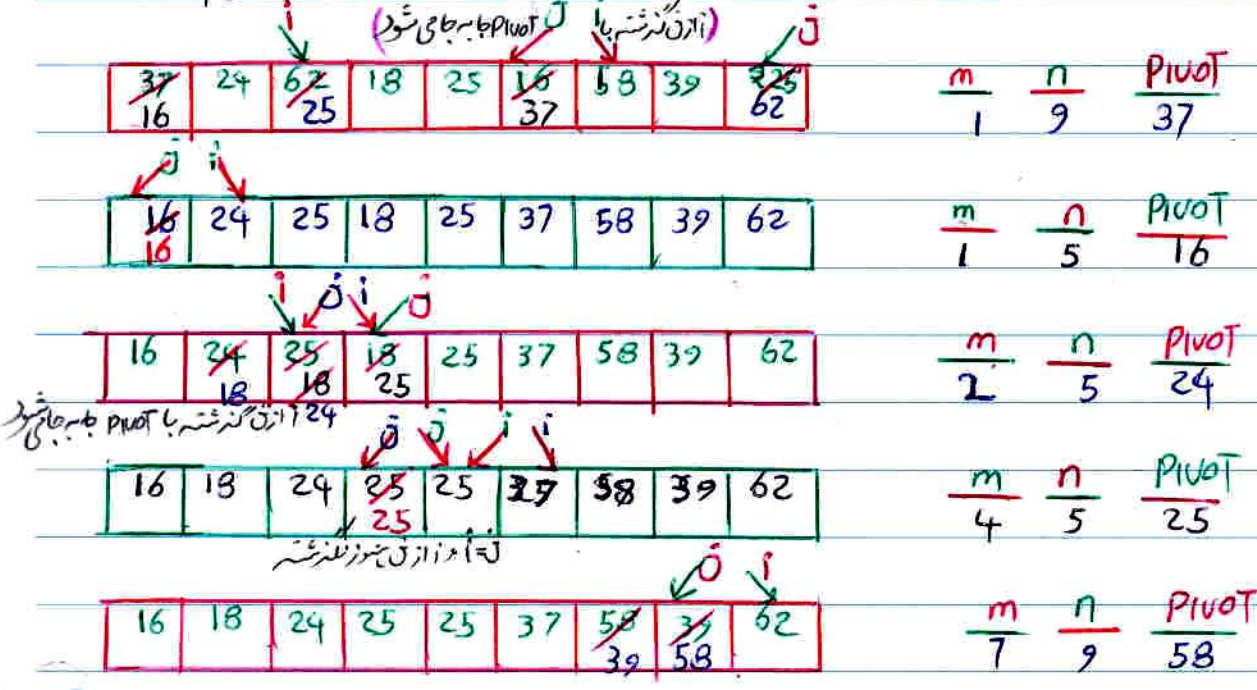
مزیت این روش نسبت به سایر روشهای مرتب سازی این است که لازم نیست همزمان تمام داده ها در حافظه اصلی موجود باشند چنین می توان با استفاده از حافظه کمکی (مثلاً دیسک سخت Hard disk) قسمتهایی را که می خواهیم در هم ادغام شود بجا یا حتی در حافظه اصلی ادغام کنیم.

زیر برنامه مرتب سازی ادغامی در صفحه ۳۷۵ آمده است.

روش مرتب سازی سریع (QUICK SORT)

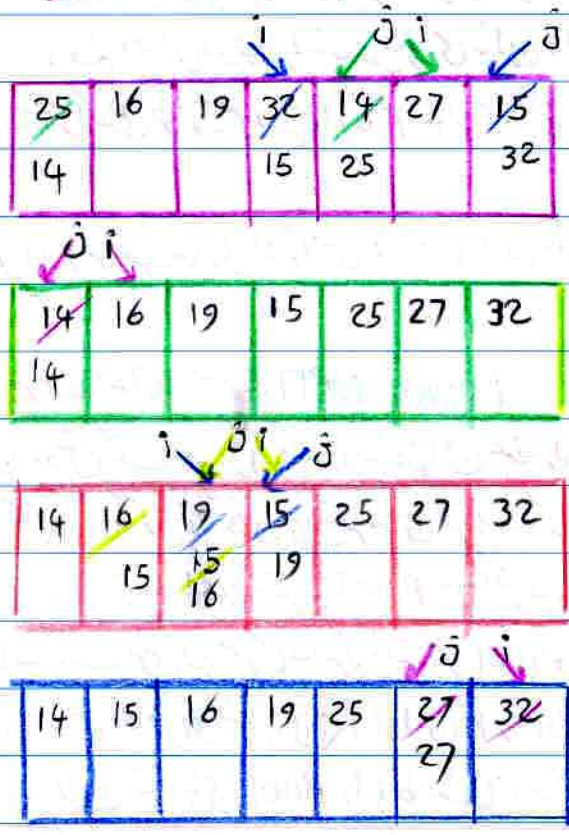
در این روش یک عنصر از آرایه را به عنوان عنصر لولا یا پاشنه گرد (PIVOT) در نظر می گیریم و مکان واقعی آن را در آرایه مشخص می کنیم بدین صورت که با متغیر i به اسم i از ابتدای آرایه شروع می کنیم و پیش می رویم تا عنصر بزرگتر از PIVOT پیدا کنیم. به همین ترتیب از انتهای آرایه با یک متغیر به نام ج خانه خانه عقب برمی گردیم تا عنصر کوچکتر از PIVOT پیدا کنیم، اگر i از ج کوچکتر باشد جای خانه ای i ام و ج ام را عوض می کنیم (ج > i) این کار را ادامه می دهیم تا i از ج بزرگتر شود (ج < i) (i از ج رد شده) در این حالت ضمناً جای PIVOT را با خانه ج ام عوض می کنیم.

پس از آنکه مکان واقعی PIVOT مشخص شد به صورت بازگشتی ابتدا به سراغ زیر آرایه سمت چپ PIVOT می رویم و سپس زیر آرایه سمت راست PIVOT عملیات را آنقدر ادامه می دهیم تا کل آرایه مرتب شود. در اینجا PIVOT را اولین خانه آرایه ای که می خواهیم مرتب شود در نظر می گیریم.



مثال :

25 ,16 ,19 ,32 , 14, 27 ,15



m	n	PIVOT
1	7	25

m	n	PIVOT
1	4	4

m	n	PIVOT
2	4	16

m	n	PIVOT
6	7	27

زیر برنامه مرتب سازی سریع در صفحه ۳۵۸ آمده است.

پروژه ۵

عنوان پروژه : تبدیل عبارات میانوندی به پسوندی

هدف پروژه : نوشتن برنامه‌ای به زبان پاسکال (یا C) که در آن ورودی یک عبارت میانوندی است با استفاده از پشته به عبارت پسوندی تبدیل شود، عبارات ورودی در واقع عبارات ریاضی هستند که از تعدادی عملوند و عملگر تشکیل شده‌اند، عملگرها عبارتند از :

+	برای جمع	**	توان
-	برای تفریق	(	پرانتز بسته
*	برای ضرب	)	پرانتز باز
/	برای تقسیم	~	منفی یکتایی (shift+1=~)

عملوندها حروف بزرگ انگلیسی‌ای باشند فرض می‌شود که هر عملوندی تنها یک حرف دارد، ضمناً تعداد پرانتزهای تو در تو حداکثر سه لایه است، پایان عبارت ورودی با # یا enter مشخص می‌شود.
(حتماً باید از پشته استفاده شود)

ورودی : A*((B-C)/(D*~E))+F#

خروجی : ABC-DE~*/*F+

مهلت تحویل
پنج شنبه ۱۵ بهمن ساعت ۱۴ بعدازظهر

آرایه (Arrays)

هر آرایه تعدادی خانه متوالی از حافظه است که از یک جنس باشند و هر جا که با داده‌ای مرتب سروکار داشته باشیم می‌توانیم از آرایه استفاده کنیم. مثلاً آرایه روزهای هفته دارای ۷ عنصر است که از شنبه شروع می‌شود و به جمعه ختم می‌گردد، همین طور آرایه سالهای تحصیل یک فرد در دانشگاه.

آرایه ساختمان داده‌ای دارند که در صفحه ۵۷ کتاب آمده است.

هر آرایه در واقع یک مجموعه از زوج‌های مرتب به صورت (مقدار، اندیس) می‌باشد مثلاً در مورد روزهای هفته آرایه به صورت زیر است:

(جمعه، ۷)، ... ، (یکشنبه، ۲)، (شنبه، ۱)

توابع اصلی آرایه

سه تابع اصلی در ساختمان داده هر آرایه حضور دارند یکی CREATE برای ایجاد، دیگری STORE برای ذخیره‌سازی آرایه و سومی RETRIEVE برای بازیابی عناصر آرایه است.

ص ۵۷

ساختمان داده آرایه Structure Array is

مجموعه‌ای از زوج‌هایی که برای هر مقدار اندیس یک مقدار از مجموعه عناصر (item) در نظر گرفته می‌شود: Objects: اندیس مجموعه متناهی مرتب از یک بعد یا بیشتر مثلاً یک مجموعه اندیس یک بعدی است $\{1, 2, \ldots, n\}$ و یک مجموعه اندیس دو بعدی است $\{(1,1), (1,2), (2,1), (2,2), \ldots\}$

توابع Functions

تابع create خروجی آرایه

یک آرایه بعد از اعلان می‌کند برد بعد از آن توسط عنصر از لیست داده شده است ضمناً عناصر آرایه از جنس یا نوع item هستند. j و list و item هم اند.

a: ARRAY [1..10] of INTEGER;

a: لیست، [1..10]: یک بعدی، INTEGER: item

Create(j, list, item): Array::=

تابع Retrieve

Retrieve(A, i): Item::=

اگر i یک اندیس از مجموعه A باشد در این صورت عنصر متناظر با آن اندیس i در آرایه A برگردانده می‌شود در غیر این صورت خطا.

تابع Store

Store(A, i, x): Array::= ورودی: (یک آرایه A و اندیس i و item ← x)

اگر i در مجموعه اندیس‌های A باشد در این صورت برای آرایه A زوج جدید (x, i) اضافه می‌گردد یا جایگزین می‌شود و حاصل که یک آرایه جدید است برگردانده می‌شود، در غیر این صورت Error می‌دهد.

جبر چند جمله‌ای

منظور از جبر چند جمله‌ای یعنی تعریف تعدادی چند جمله‌ای و سپس انجام محاسبات ساده ریاضی نظیر جمع، ضرب، تقسیم، مشتق گیری و ... روی چند جمله‌ای است.

یک چند جمله‌ای درجه n دارای شکل کلی زیر است:

$$\sum_{i=0}^{n} c_i x^i = c_0 x^0 + c_1 x + c_2 x^2 + \dots + c_n x^n$$

c_i ضریب و اعداد $(n, \dots, 1)$ توان می باشند.

مثال ۱

$$A(x) = 3x^7 - 5x^3 + 2x - 4$$

یک چند جمله‌ای درجه ۷ می باشد، در واقع چند جمله‌ای $A(x)$ را می توان به صورت زیر نمایش داد:

$$3x^7 + 0x^6 + 0x^5 + 0x^4 + (-5)x^3 + 0x^2 + 2x + (-4)$$

یک راه مناسب

یک راه مناسب برای نمایش این چند جمله‌ای استفاده از آرایه است، می توانیم آرایه‌ای ۸ عنصری تعریف کنیم و ضرایب چند جمله‌ای $A(x)$ را در آن آرایه قرار دهیم. با این حساب چند جمله‌ای $A(x)$ در آرایه a به صورت زیر ذخیره می شود:

0	1	2	3	4	5	6	7
a[0] →	3	0	0	0	-5	0	2	-4

برای شمارش جمع

اگر بخواهیم $A(x)$ را با یک چند جمله‌ای دیگر مانند $B(x)$

$$B(x) = x^9 - 2x^7 + 5x^3 + x^2$$

جمع کنیم باید $B(x)$ را در یک آرایه ۱۰ عنصری به نام b به صورت زیر ذخیره کنیم:

0	1	2	3	4	5	6	7	8	9
b →	1	0	-2	0	0	0	5	1	0	0

سپس برای شمارش جمع $A(x)+B(x)$ باید ابتدا در خانه صفر در سمت چپ آرایه a قرار دهیم تا دو چند جمله‌ای هم درجه شود، سپس با استفاده از قطعه برنامه زیر می توانیم حاصل دو چند جمله‌ای را داخل آرایه قرار دهیم:

```
FOR i := 0 TO 9 DO
    c[i] := a[i] + b[i];
```

تذکر: در حالت کلی اگر $A(x)$ درجه m و $B(x)$ درجه n باشد، زمان اجرای الگوریتم فوق به صورت $O(\max(m,n))$ خواهد بود.

روش قبل اولین روشی است که به ذهن می رسد اما کارایی چندانی ندارد، مثلاً فرض کنید بخواهیم دو چند جمله‌ای $P(x)=2x^{1000}+5$ و $Q(x)=x^2-3$ را با هم جمع کنیم اولاً برای ذخیره سازی $P(x)$ به آرایه‌ای به طول ۱۰۰۱ نیاز داریم که ۹۹۹ خانه آن صفر است ضمناً برای جمع $Q(x)$ و $P(x)$ باید سمت چپ $Q(x)$ ۹۹۸ صفر قرار دهیم تا هم درجه شود، با توجه به این مشکلات این روش نه از لحاظ حافظه مطلوب است و نه از لحاظ سرعت اجرا زیرا عملاً باید از حلقه به طول ۱۰۰۱ جهت جمع چند جمله‌ای استفاده کنیم.

به همین خاطر همانطور که در ریاضیات از نوشتن جملات با ضریب صفر خودداری می‌کنیم، در جبر چند جمله‌ای نیز جملات صفر را ذخیره نمی‌کنیم، اگر بخواهیم تنها جملات غیر صفر را ذخیره کنیم همراه با ضریب (COEFICIENT) باید حتماً توان یا نما (EXPONENT) را نیز ذخیره کنیم.

COEF
EXP

به همین خاطر هر جمله، یک چند جمله‌ای یک ساختار دو عضوی خواهد بود چنین ساختاری را در زبان پاسکال می‌توانیم به هنگام تعریف نوع داده (Data Type) تعریف کنیم، در واقع هر جمله (Term) یک چند جمله‌ای ساختار زیر را دارد:

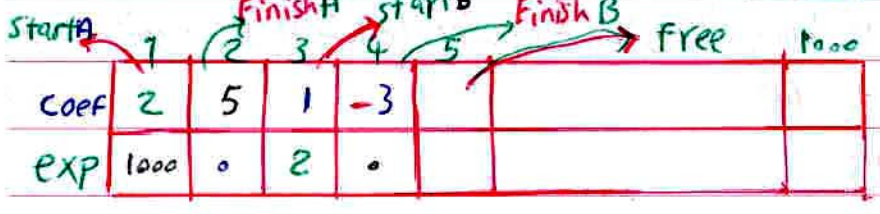
```
TYPE term=RECORD
     coef: REAL;
     exp: 0..MAXINT;
     END;
```

حروف بزرگ: کلمات کلیدی
حروف کوچک: متغیر

برای تعریف یک چند جمله‌ای می‌توانیم در قسمت متغیرها (VAR) چند جمله‌ای terms را به صورت زیر تعریف کنیم:

```
VAR terms: ARRAY[1..1000] of term;
```

برای ذخیره ساختن مقدار در این ساختمان داده (همان تابع) STORE با استفاده از عملگر dot انتخاب ۱۰۱ مثلاً اگر بخواهیم جمله اول terms را $2x^{1000}$ قرار دهیم در قسمت coef عدد ۲ و در قسمت exp عدد ۱۰۰۰ را قرار می‌دهیم.




```
terms[1].coef := 2;
terms[1].exp := 1000;
```

تمام جملات چند جمله ای های A(X) و B(X) را می توانیم به صورت زیر در آرایه عمومی terms قرار دهیم برای این منظور باید خانه شروع چند جمله ای A را start A و پایان چند جمله ای A را Finish A، همچنین شروع و پایان چند جمله ای B(X) تعیین شود ضمناً اولین خانه خالی چند جمله ای terms باید با یک متغیر سراسری به نام Free مشخص باشد.

8

"81, 7, 22

• سیمرتوالی •

جمع چند جمله ای ها (Padd)

در قسمت قبل دیدیم که چگونه می توان یک چند جمله ای را به صورت فشرده یعنی بدون در نظر گرفتن جملات صفر ذخیره کرد. در اینجا طی الگوریتم Padd حاصل جمع چند جمله ای های A(X) و B(X) را تعیین می کنیم برای این منظور باید روی تک تک جملات چند جمله ای A(X) و همین طور B(X) حرکت کنیم، از دو متغیر a و b استفاده می کنیم متغیر a روی جملات A(X) حرکت می کند یعنی محدوده تغییرات آن از start A تا Finish A می باشد به همین ترتیب متغیر b هر بار قسمت توان جمله ای را که a به آن اشاره می کند با قسمت توان جمله ای را که b به آن اشاره می کند مقایسه می کنیم (این مقایسه توسط دستور COMPARE (terms[a].exp , terms[b].exp) انجام می شود.)

سه حالت ممکن است رخ دهد:

الف) terms(a).exp > terms(b).exp

در این حالت جمله چند جمله ای A(X) توانی بزرگتر از چند جمله ای B(X) دارد پس یا یکی از جمله A(X) را در چند جمله ای حاصل جمع که آن را C(X) می نامیم بنویسیم (عمل نوشتن جمله جدید در چند جمله ای C(X) توسط زیر برنامه NewTerm انجام می شود) در نهایت پس از نوشتن جمله A(X) در C(X) یک واحد به a اضافه می کنیم زیرا تکلیف آن جمله مشخص شود.

ب) حالت مساوی terms(a).exp = terms(b).exp

در این حالت جمله های A(X) و B(X) توان مساوی دارند پس باید ضرایب آنها با هم جمع شود اگر مجموع ضرایب مخالف صفر بود آنگاه حاصل جمع را به همراه یکی از توانها در چند جمله ای C(X) می نویسیم، اما اگر مجموع ضرایب صفر شد در C(X) چیزی نمی نویسیم، در نهایت چه مجموع ضرایب صفر باشد و چه غیر صفر یک واحد به a اضافه می کنیم و هم یک واحد به b.