به نام خدا

ساختمان داده ها

جلسه دوم

دانشگاه صنعتی همدان

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

آرایه ها

- ADT ارایه
- نمایش ارایه ها
- ADT چند جمله ایها
- ADTماتریسهای خلوت

آرایه

آرایه‌ها Array

آرایه نوعی ساختمان داده است که عناصر آن هم نوع بوده و هر یک از عناصر با یک اندیس به صورت مستقیم قابل دستیابی است. آرایه می‌تواند یک بعدی ، دو بعدی و یا چند بعدی باشد. آرایه‌های دو بعدی را با نام ماتریس می‌شناسیم.

Objects: *A set of pairs <index, value> where for each value of index there is a value from the set item. Index is a finite ordered set of one or more dimensions, for example, {0, ... , n-1} for one dimension, {(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)} for two dimensions, etc.*

Methods:

for all A ∈ Array, i ∈ index, x ∈ item, j, size ∈ integer

*Array Create(j, list) ::= **return** an array of j dimensions where list is a j-tuple whose kth element is the size of the kth dimension. Items are undefined.*

*Item Retrieve(A, i) ::= **if** (i ∈ index) **return** the item associated with index value i in array A*
*else **return** error*

*Array Store(A, i, x) ::= **if** (i in index)*
*__return__ an array that is identical to array A except the new pair <i, x> has been inserted **else return** error*

نمایش آرایه ها

- ارایه های یک بعدی به صورت خانه های پشت سرهم در حافظه قرار می گیرند. بنابراین برای یک آرایه مثل

Item Array[n]

n= تعداد عناصر ارایه

فضای اشغال شده=n*sizeof(item)

اگر ارایه از ادرس α شروع شده باشد انگاه

Array[i]ادرس=α +i*sizeof(item)

- ارایه های چند بعدی نیز به صورت ارایه های یک بعدی پیاده سازی می شوند. برای این کار دوروش وجود دارد. روش سطری و روش ستونی.

$$\begin{bmatrix} 2 & 5 \\ 1 & 6 \\ 3 & 4 \end{bmatrix} \quad 3\times 2$$

۱. روش سطری Row Major

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

سطری

۲. روش ستونی Column Major

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

ستونی

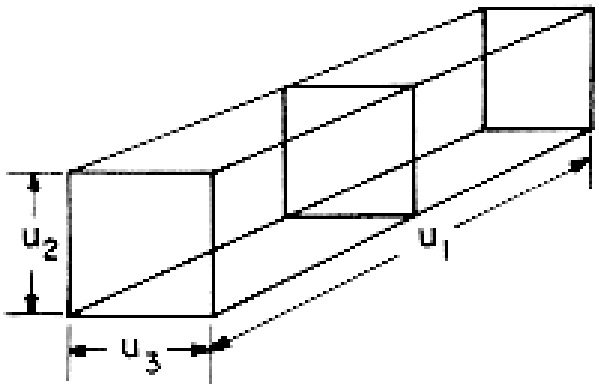
- ادرس عناصر ارایه Array[u1][u2] در روش سطری

Array[i,j]= α +i*u2+j

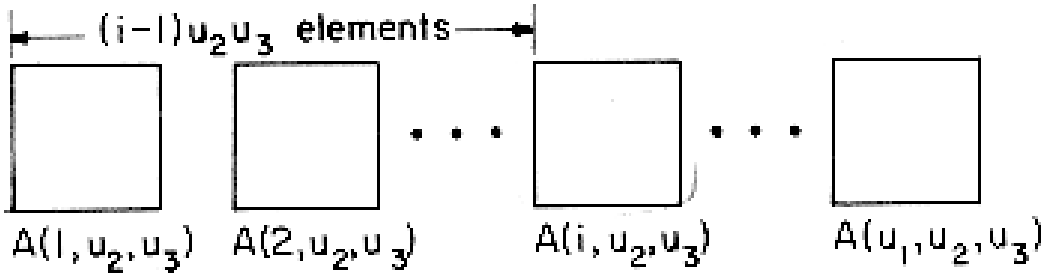
- در روش ستونی:

Array[I,j]= α +j*u1+i

نمایش ارایه های چند بعدی



(a) 3-dimensional array $A(u_1,u_2,u_3)$ regarded as u_1 2-dimensional arrays.



آرایه های چند بعدی

- ارایه سه بعدی ادرس:

$$Array[i,j,k]= \alpha +i*u_2*u_3+j*u_3+k$$

در حالت کلی برای یک ارایه n بعدی با ابعاد $u_1,u_2,\ldots u_n$

$$Array[i_1,i_2,i_3,..i_n]=\alpha+i_1*u_2*u_3..*u_n+i_2*u_3*\ldots*u_n+\ldots+i_n$$

$$= \alpha + \sum_{j=1}^{n} i_j \quad a_j \quad \text{with} \begin{cases} a_j = \prod_{k=j+1}^{n} u_k & 1 \leqq j < n \\ a_n = 1 \end{cases}$$

ماتریسهای مثلثی

در آرایـه‌هـای دو بعـدی مربعی یا ماتریس‌های مربعی که کلیه عناصر بالای قطر اصلی آن صفر باشند یـک ماتریس پایین مثلثی تشکیل می‌گردد و برعکس اگر کلیه عناصر پایین قطر اصلی آن صفر باشند یـک ماتـریس بـالا مثلثـی تشـکیل خـواهد شـد. در یـک ماتـریس پایین مثلثی یا بالا مثلثی حداکثر $\frac{n(n+1)}{2}$ عنصر غیر صفر داریم که n اندازه هر بعد ماتریس است.

$$\begin{bmatrix} 1 & 6 & 7 \\ 0 & 2 & 5 \\ 0 & 0 & 4 \end{bmatrix}$$

بالا مثلثی

حداکثر عناصر غیر صفر $= \frac{3(3+1)}{2} = 6$

$A[i, j] = 0 \quad i > j \Longrightarrow$ ماتریس بالا مثلثی

$A[i, j] = 0 \quad i < j \Longrightarrow$ ماتریس پایین مثلثی

اگـر انـدازه ابعـاد ماتریس‌های مثلثی افزایش یابند این ماتریس‌ها حاوی تعداد زیادی صفر خواهند بود کـه ذخیـره کردن سطری یا ستونی ماتریس به طور کامل در حافظه باعث هدر رفتن بخشی از فضای حافظـه مـی‌گردد. به همین دلیل ماتریس‌های مثلثی را بصورت سطری یا ستونی بدون در نظر گرفتن صفرها در حافظه ذخیره می‌کنند.

پایین مثلثی ======> سطری $\frac{(i+1)\times i}{2}+j$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 5 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

======>

0	1	2	3	4	5
1	2	5	3	1	1

پایین مثلثی

استفاده از ارایه برای پیاده سازی ADTهای دیگر- چند جمله ایها

- معمولا از آرایه که خود یک ADT است برای پیاده سازی ADT های دیگر استفاده می شود. یکی از مثالهای خوب برای این کاربرد؛ چند جمله ایها هستند. می خواهیم ADT پیاده کنیم که در ان چند جمله ایها را ذخیره کرده و یک سری اعمال روی انها انجام دهیم.

$$A(x) = 3x^2 + 2x + 4 \text{ and } B(x) = x^4 + 10x^3 + 3x^2 + 1$$

Class Polynomial{

Objects: a set of ordered pairs of $<e_i,a_i>$
where a_i in *Coefficients* and
e_i in *Exponents*, e_i are integers >= 0

Methods:

for all *poly, poly1, poly2* ∈ *Polynomial*, *coef* ∈*Coefficients*, *expon* ∈*Exponents*

Polynomial Zero( ) ::= **return** the polynomial *p(0)*

Boolean IsZero(*poly*) ::= **if** (*poly*) **return** *FALSE*
else return *TRUE*

Coefficient Coef(*poly, expon*) ::= **if** (*expon* ∈ *poly*) **return** its
coefficient **else return** Zero

Exponent Lead_Exp(*poly*) ::= **return** the largest exponent in ***poly***

Polynomial Attach(*poly,coef, expon*) ::= **if** (*expon* ∈ *poly*) **return** error
else return the polynomial poly
with the term <*coef, expon*> inserted

Polynomial Remove(*poly, expon*) ::= **if** (*expon* ∈ *poly*) **return** the polynomial *poly* with the term whose exponent is *expon deleted*
else return error

Polynomial SingleMult(*poly, coef, expon*)::= **return** the polynomial
poly • *coef* • x^{expon}

Polynomial Add(*poly1, poly2*) ::= **return** the polynomial
*poly*1 +*poly*2

Polynomial Mult(*poly1, poly2*) ::= **return** the polynomial
*poly*1 • *poly*2

}

پیاده سازی ADT چند جمله ای ها

- نمایش چند جمله ایها

```
class term {
friend Polynomial;
private:
float  coef;
Int  exp;
}

class Polynomial{
private:
static term termArray[Max];
static int free;
int start, finish;
```

Store pairs of exponent and coefficient

$A(X)=2X^{1000}+1$

$B(X)=X^4+10X^3+3X^2+1$

advantage: less space

disadvantage: longer code