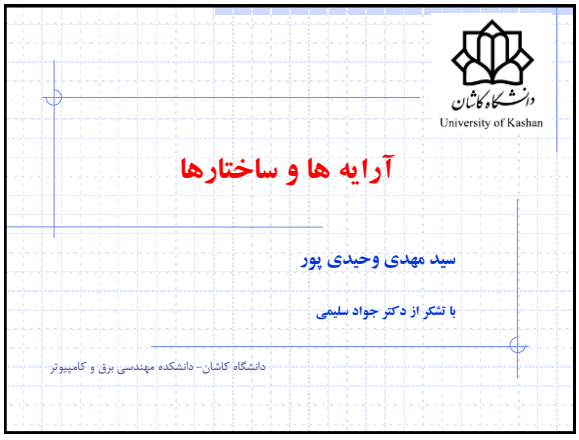
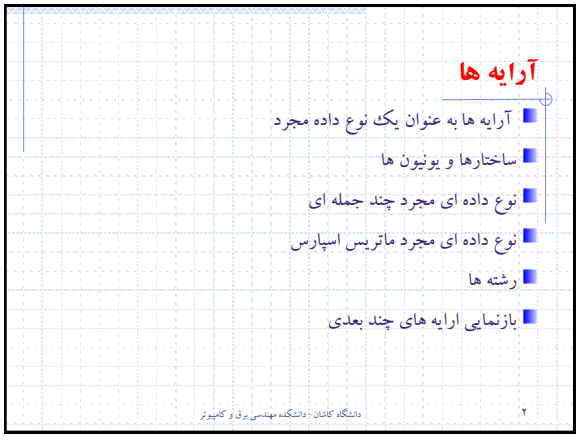
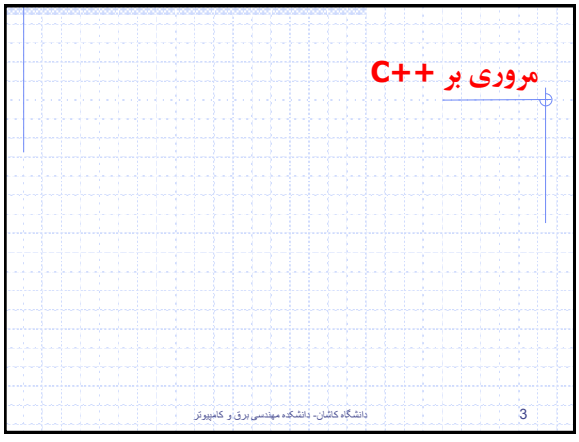

دانشگاه کاشان
University of Kashan
آرایه ها و ساختارها
سید مهدی وحیدی پور
با تشکر از دکتر جواد سلیمی
دانشگاه کاشان- دانشکده مهندسی برق و کامپیوتر


آرایه ها
آرایه ها به عنوان یک نوع داده مجرد
ساختارها و یونیون ها
نوع داده ای مجرد چند جمله ای
نوع داده ای مجرد ماتریس اسپارس
رشته ها
بازنمایی ارایه های چند بعدی
دانشگاه کاشان - دانشکده مهندسی برق و کامپیوتر
۲


مروری بر C++
دانشگاه کاشان- دانشکده مهندسی برق و کامپیوتر
3

روشهای دسته بندی داده ها در کنار هم

آرایه	• مجموعه ای از عناصر از یک نوع داده می باشد
ساختار	• مجموعه ای از عناصر است که لزومی ندارد داده های آن یکسان باشد
یونیون	• اعلان یونیون مشابه تعریف و اعلان یک ساختار است با این تفاوت که فیلدهای یک یونیون باید در حافظه با هم مشترک باشند
کلاس	• مجموعه ای از عناصر است که لزومی ندارد داده های آن یکسان باشد به همراه توابعی که بر روی آنها اجرا می‌شوند

ساختارها

```
struct {
        char name[10];
        int age;
        float salary;
        } person;
```

```
strcpy(person.name,"james");
person.age = 10;
person.salary = 35000;
```

- ما می توانیم ساختارهای داده خود را به کمک typedef ایجاد کنیم

```
typedef struct human_being {
        char name[10];
        int age;
        float salary;
        };
```

or

```
typedef struct {
        char name[10];
        int age;
        float salary;
        } human_being;
```

- حال می توان از این نوع داده ای استفاده کرد

human_being ali, hasan;

یونیون

- فقط یک فیلد یونیون در هر زمان فعال می گردد.

```
enum   kind  {stu,emp};
typedef  struct  person {
          kind   flag;
          union   {
              int   empno;
              long int  stuno;
          } no;
};
```

```
person   ali,hasan;

ali.flag= stu;
Ali.no.stuno=13245346753;

hasan.flag=emp;
hasan.no.empno=14353;
```

نوع داده ای مجرد

نوع داده مجرد یا انتزاعی

- نوع داده مجرد یا انتزاعی (ADT) نوع داده ای است که در ان مشخصات داده‌ها و اعمال بر روی آنها از بازنمایی و پیاده سازی داده جدا می شود
- پنهان سازی اطلاعات: مخفی کردن جزییات پیاده سازی اشیا داده ای از دید دنیای خارج

Abstracted Data Type

آرایه به عنوان یک نوع داده مجرد

structure *Array* is

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.

functions:

for all $A \in Array, i \in index, x \in item, j, size \in$ integer

Array Create(*j*, *list*)	::=	**return** an array of *j* dimensions where *list* is a *j*-tuple whose *i*th element is the the size of the *i*th 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.

آرایه مجموعه ای از زوج ها، شامل اندیس و مقدار است

end *Array*

لیست ها

- لیست های مرتب شده یا خطی به صورت
- Ordered (linear) list: (item1, item2, item3, …, item*n*)
- مثال
  - (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday)
  - (Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King)
  - (1941, 1942, 1943, 1944, 1945)
  - (a_1, a_2, a_3, …, a_{n-1}, a_n)

لیست ها

- اعمال
  1. پیدا کردن طول یک لیست
  2. خواندن اقلام داده یک لیست از چپ به راست یا بر عکس
  3. بازیابی i امین عنصر از یک لیست
  4. تعویض یک قلم اطلاعاتی در i امین موقعیت یک لیست
  5. درج یک قلم داده جدید در i امین موقعیت یک لیست
  6. حذف یک قلم اطلاعاتی از i امین موقعیت یک لیست

نمایش یک لیست مرتب شده به صورت یک آرایه

- پیاده سازی
  - ✓ نگاشت ترتیبی **sequential mapping (1)~(4)**
  - ✓ نگاشت غیر ترتیبی **non-sequential mapping (5)~(6)**

نوع داده ای مجرد چند جمله ای ۱

structure *Polynomial* is
objects: $p(x) = a_1x^{e_1} + \cdots + a_nx^{e_n}$; a set of ordered pairs of $\langle e_i, a_i \rangle$ where a_i in *Coefficients* and e_i in *Exponents*, e_i are integers >= 0
functions:
for all *poly*, *poly1*, *poly2* ∈ *Polynomial*, *coef* ∈ *Coefficients*, *expon* ∈ *Exponents*

Polynomial Zero()	::=	**return** the polynomial, $p(x) = 0$
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 $poly \cdot coef \cdot x^{expon}$
Polynomial Add(*poly1*, *poly2*)	::=	**return** the polynomial $poly1 + poly2$
Polynomial Mult(*poly1*, *poly2*)	::=	**return** the polynomial $poly1 \cdot poly2$

end *Polynomial*

نوع داده ای مجرد چند جمله ای ۲

- چند جمله ایهای نمونه

$A(x) = 3x^{20}+2x^5+4$ and $B(x) = x^4+10x^3+3x^2+1$

- فرض کنید دو چندجمله ای زیر را داشته باشیم که در انها X متغیر و a_i ضریب و i توان است آنگاه

$$A(x) + B(x) = \Sigma(a^i + b^i)x^i$$
$$A(x) \cdot B(x) = \Sigma(a^ix^i \cdot \Sigma(b^ix^i))$$

- به صورت مشابه می‌توان تفریق و تقسیم چندجمله ای‌ها و بسیاری از عملیات دیگر را تعریف کرد.

نوع داده ای مجرد چند جمله ای ۳

- Representation I

a.degree=n
a.coef[i]=a_{n-i}

ضرایب به ترتیب نزول درجه در ارایه ذخیره شود

```
#define MAX_degree 101
/*MAX degree of polynomial+1*/
typedef struct{
  int degree;
  float coef [MAX_degree];
}polynomial;
```

Drawback: The first representation may waste space.

نوع داده ای مجرد چند جمله ای ٤

- Representation II

آرایه coef را به گونه ای در نظر بگیریم که طول آن برابر a.degree+1 شود

```
Class polynomial{
  private:
    int degree;
    float *coef;
  };
Polynomial::polynomial( int d)
{
   degree=d;
   coef=new float[degree+1]
}
```

Drawback: $X^{1000}+1$ in this representation has 2 nonzero term

نوع داده ای مجرد چند جمله ای ٥

- Representation III
  - تمام چند جمله ایها را در یک ارایه ذخیره کنیم
  - فقط ضرایب غیر صفر را به همراه توان انها ذخیره کنیم

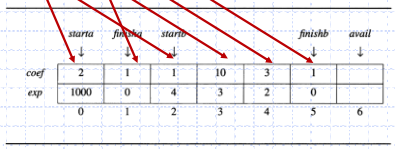
```
#define MAX_TERMS 100
/*size of terms array*/
typedef struct{
  float coef;
  int expon;
}polynomial;

polynomial terms [MAX_TERMS];
int avail = 0;
```

نوع داده ای مجرد چند جمله ای ۶

storage requirements: start, finish, 2*(finish-start+1)

representation <start, finish>
specification poly
<0,1> A
<2,5> B

$A(x) = 2x^{1000}+1$
$B(x) = x^4+10x^3+3x^2+1$



نوع داده ای مجرد چند جمله ای ۷

```
void padd(int starta,int finisha,int startb, int finishb,
                                int *startd,int *finishd)
{
/* add A(x) and B(x) to obtain D(x) */
  float coefficient;
  *startd = avail;
  while (starta <= finisha && startb <= finishb)
    switch(COMPARE(terms[starta].expon,
                   terms[startb].expon)) {
      case -1: /* a expon < b expon */
            attach(terms[startb].coef,terms[startb].expon)
            startb++;
            break;
      case 0: /* equal exponents */
            coefficient = terms[starta].coef +
                          terms[startb].coef;
            if (coefficient)
              attach(coefficient,terms[starta].expon);
            starta++;
            startb++;
            break;
      case 1: /* a expon > b expon */
            attach(terms[starta].coef,terms[starta].expon)
            starta++;
  }
  /* add in remaining terms of A(x) */
  for(; starta <= finisha; starta++)
    attach(terms[starta].coef,terms[starta].expon);
  /* add in remaining terms of B(x) */
  for( ; startb <= finishb; startb++)
    attach(terms[startb].coef, terms[startb].expon);
  *finishd = avail-1;
}
```

یک تابع c که دو چند جمله ای A و B را جمع میکند تا D را به دست آورد $D = A + B$

Analysis: $O(n+m)$ where n (m) is the number of nonzeros in A (B).

نوع داده ای مجرد چند جمله ای ۸

```
void attach(float coefficient, int exponent)
{
/* add a new term to the polynomial */
  if (avail >= MAX_TERMS) {
    fprintf(stderr,"Too many terms in the polynomial\n");
    exit(1);
  }
  terms[avail].coef = coefficient;
  terms[avail++].expon = exponent;
}
```

مشکل: هنگامی که چندجمله ای لازم نباشد باید فشرده سازی انجام شود
جابجایی داده انجام می شود

ماتریس اسپارس

- در ریاضیات ، یک ماتریس شامل m سطر و n ستون از اعضا می باشد
- ماتریسی که عناصر صفر آن زیاد بوده ماتریس اسپارس نامیده می شود.

	col 0	col 1	col 2
row 0	−27	3	4
row 1	6	82	−2
row 2	109	−64	11
row 3	12	8	9
row 4	48	27	47

5*3

(a) 15/15

	col 0	col 1	col 2	col 3	col 4	col 5
row 0	15	0	0	22	0	−15
row 1	0	11	3	0	0	0
row 2	0	0	0	−6	0	0
row 3	0	0	0	0	0	0
row 4	91	0	0	0	0	0
row 5	0	0	28	0	0	0

6*6

(b) 8/36

sparse matrix data structure?

ماتریس اسپارس

structure *Sparse_Matrix* is
objects: a set of triples, <*row*, *column*, *value*>, where *row* and *column* are integers and form a unique combination, and *value* comes from the set *item*.
functions:
for all $a, b \in Sparse\_Matrix$, $x \in item$, $i, j, max\_col, max\_row \in index$

Sparse_Matrix Create(*max_row*, *max_col*) ::=
return a *Sparse_Matrix* that can hold up to *max_items* = *max_row* × *max_col* and whose maximum row size is *max_row* and whose maximum column size is *max_col*.

Sparse_Matrix Transpose(*a*) ::=
return the matrix produced by interchanging the row and column value of every triple.

Sparse_Matrix Add(*a*, *b*) ::=
if the dimensions of *a* and *b* are the same
return the matrix produced by adding corresponding items, namely those with identical *row* and *column* values.
else return error

Sparse_Matrix Multiply(*a*, *b*) ::=
if number of columns in *a* equals number of rows in *b*
return the matrix *d* produced by multiplying *a* by *b* according to the formula: $d[i][j] = \sum(a[i][k] \cdot b[k][j])$ where $d(i, j)$ is the (i, j)th element
else return error.

حداقل اعمال ممکن شامل ایجاد، جمع ، ضرب و ترانهاده ماتریس می باشد.

۲۱

ماتریس اسپارس

- می توان از یک آرایه از سه تایی های <row, col, value> برای نمایش یک ماتریس اسپارس استفاده کرد.

row, column in ascending order

# of nonzero terms

# of rows (columns)

	col 0	col 1	col 2	col 3	col 4	col 5
row 0	15	0	0	22	0	−15
row 1	0	11	3	0	0	0
row 2	0	0	0	−6	0	0
row 3	0	0	0	0	0	0
row 4	91	0	0	0	0	0
row 5	0	0	28	0	0	0

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

(a)

ماتریس اسپارس

- پیاده سازی *Create* operation

Sparse_Matrix Create(*max_row, max_col*) ::=

```
#define MAX_TERMS 101 /* maximum number of terms +1*/
typedef struct {
        int col;
        int row;
        int value;
        } term;
term a[MAX_TERMS];
```


ترانهاده یک ماتریس اسپارس

- برای پیدا نمودن ترانهاده یک ماتریس باید جای سطرها و ستون ها را عوض کرد بدین مفهوم که هر عنصر a[i][j] در ماتریس اولیه به عنصر b[j][i] در ماتریس ترانهاده تبدیل می شود.

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

(a)

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]	2	5	28
[6]	3	0	22
[7]	3	2	−6
[8]	5	0	−15

(b)

ترانهاده یک ماتریس اسپارس

- For each row i
  - take element <i, j, value> and store it in element <j, i, value> of the transpose.
  - difficulty: where to put <j, i, value>

    (0, 0, 15) ====> (0, 0, 15)
    (0, 3, 22) ====> (3, 0, 22)
    (0, 5, -15) ====> (5, 0, -15)
    (1, 1, 11) ====> (1, 1, 11)

- For all elements in column j,
  - place element <i, j, value> in element <j, i, value>

- الگوریتم بیان شده نشان می دهد که باید تمام عناصر در ستون ۰ را پیدا و آنها را در سطر ۰ ذخیره کرد همچنین تمام عناصر ستون ۱ را پیدا و در سطر ۱ قرار داد و همین فرآیند را ادامه داد. از آنجا که ماتریس اولیه سطری بوده لذا ستون های داخل هر سطر از ماتریس ترانهاده نیز به صورت صعودی مرتب می شود.

الگوریتم ترانهاده

Assign
A[i][j] to B[j][i]

place element <i, j, value>
in element <j, i, value>

For all columns i
For all elements in column j

Scan the array
"columns" times.
The array has
"elements" elements.

```
void transpose(term a[], term b[])
/* b is set to the transpose of a */
{
  int n,i,j, currentb;
  n = a[0].value;       /* total number of elements */
  b[0].row = a[0].col; /* rows in b = columns in a */
  b[0].col = a[0].row; /* columns in b = rows in a */
  b[0].value = n;
  if (n > 0 )  { /* non zero matrix */
    currentb = 1;
    for (i = 0; i < a[0].col; i++)
    /* transpose by the columns in a */
      for (j = 1; j <= n; j++)
      /* find elements from the current column */
        if (a[j].col == i) {
        /* element is in current column, add it to b */
          b[currentb].row = a[j].col;
          b[currentb].col = a[j].row;
          b[currentb].value = a[j].value;
          currentb++;
        }
  }
}
```

==> O(columns*elements)

EX: A[6][6] transpose to B[6][6]

i=1 j=8
a[i].col = 2 != i

Matrix A

	Row	Col	Value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

Row Col Value

0	6	6	8
1	0	0	15
2	0	4	91
3	1	1	11

Set Up row & column in B[6][6]

And So on...

بحث در مورد الگوریتم ترانهاده

- مقایسه الگوریتم ارائه شده برای بازنمایی اسپارس و بازنمایی دوبعدی
  - O(columns*elements) vs. O(columns*rows)
  - elements --> columns * rows وقتی ماتریس اسپارس نباشد O(columns²*rows)
- مشکل: ارایه columns بار بررسی می شود
  - می توان ترانهاده ماتریسی که به صورت سه تاییها نگهداری شده را در زمان O(columns + elements) پیدا کرد.
- راهکار:
  - تعداد عناصر در هر ستون ماتریس اولیه را مشخص کنید
  - شروع هر سطر ماتریس ترانهاده را تعیین کنید

الگوریتم ترانهاده سریع

- تعداد عضوهای هر ستون ماتریس A تعداد عضوهای هر سطرB را به دست می دهد
- Rowstart[i] برابر RowStart[i-1]+RowSize[i-1] است

	[0]	[1]	[2]	[3]	[4]	[5]
row_terms =	2	1	2	2	0	1
starting_pos =	1	3	4	6	8	8

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

(a)

transpose

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]	2	5	28
[6]	3	0	22
[7]	3	2	−6
[8]	5	0	−15

(b)

	[0]	[1]	[2]	[3]	[4]	[5]	
row_terms	2	1	2	2	0	1	#col = 6
starting_pos	1	3	4	6	8	8	#term = 6

Matrix A

	Row	Col	Value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

```
void fast_transpose(term a[], term b[])
{
/* the transpose of a is placed in b */
  int row_terms[MAX_COL], starting_pos[MAX_COL];
  int i,j, num_cols = a[0].col, num_terms = a[0].value;
  b[0].row = num_cols;  b[0].col = a[0].row;
  b[0].value = num_terms;
  if (num_terms > 0) { /* nonzero matrix */
    for (i = 0; i < num_cols; i++)
      row_terms[i] = 0;
    for (i = 1; i <= num_terms; i++)
      row_terms[a[i].col]++;
    starting_pos[0] = 1;
    for (i = 1; i < num_cols; i++)
      starting_pos[i] =
                starting_pos[i-1] + row_terms[i-1];
    for (i = 1; i <= num_terms; i++) {
      j = starting_pos[a[i].col]++;
      b[j].row = a[i].col;   b[j].col = a[i].row;
      b[j].value = a[i].value;
    }
  }
}
```

I = 8

	[0]	[1]	[2]	[3]	[4]	[5]
row_terms =	2	1	2	2	0	1
starting_pos =	3	4	6	8	8	9

Matrix A

	Row	Col	Value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

	Row	Col	Value
0	6	6	8
1	0	0	15
2	0	4	91
3	1	1	11
4	2	1	3
5	2	5	28
6	3	0	22
7	3	2	-6
8	5	0	-15

```
void fast_transpose(term a[], term b[])
{
/* the transpose of a is placed in b */
  int row_terms[MAX_COL], starting_pos[MAX_COL];
  int i,j, num_cols = a[0].col, num_terms = a[0].value;
  b[0].row = num_cols;  b[0].col = a[0].row;
  b[0].value = num_terms;
  if (num_terms > 0) { /* nonzero matrix */
    for (i = 0; i < num_cols; i++)
      row_terms[i] = 0;
    for (i = 1; i <= num_terms; i++)
      row_terms[a[i].col]++;
    starting_pos[0] = 1;
    for (i = 1; i < num_cols; i++)
      starting_pos[i] =
                starting_pos[i-1] + row_terms[i-1];
    for (i = 1; i <= num_terms; i++) {
      j = starting_pos[a[i].col]++;
      b[j].row = a[i].col;   b[j].col = a[i].row;
      b[j].value = a[i].value;
    }
  }
}
```

بحث در مورد الگوریتم ترانهاده سریع

Buildup
row_term
& starting_pos

transpose

For columns
For elements
For columns
For elements

```
void fast_transpose(term a[], term b[])
{
/* the transpose of a is placed in b */
  int row_terms[MAX_COL], starting_pos[MAX_COL];
  int i,j, num_cols = a[0].col, num_terms = a[0].value;
  b[0].row = num_cols;  b[0].col = a[0].row;
  b[0].value = num_terms;
  if (num_terms > 0) { /* nonzero matrix */
    for (i = 0; i < num_cols; i++)
      row_terms[i] = 0;
    for (i = 1; i <= num_terms; i++)
      row_terms[a[i].col]++;
    starting_pos[0] = 1;
    for (i = 1; i < num_cols; i++)
      starting_pos[i] =
                starting_pos[i-1] + row_terms[i-1];
    for (i = 1; i <= num_terms; i++) {
      j = starting_pos[a[i].col]++;
      b[j].row = a[i].col;   b[j].col = a[i].row;
      b[j].value = a[i].value;
    }
  }
}
```

بحث در مورد الگوریتم ترانهاده سریع

- مقایسه الگوریتم ارائه شده برای بازنمایی اسپارس و بازنمایی دوبعدی وقتی ماتریس اسپارس باشد
  - O(columns+elements) vs. O(columns*rows)
    هم در حافظه و هم در زمان اجرا صرفه جویی خواهد شد
  - elements --> columns * rows وقتی ماتریس اسپارس نباشد
    O(columns*rows)

شبیه حالت استفاده از بازنمایی دوبعدی، فقط ضریب ثابت FastTranspose بزرگتر از حالت ارایه ای

- هزینه: FastTranspose در مقایسه با transpose نیازمند حافظه بیشتری می باشد.

Additional row_terms and starting_pos arrays

واهکار: از همان حافظه مشترک برای نمایش دو ارایه row_terms و starting_pos استفاده شود

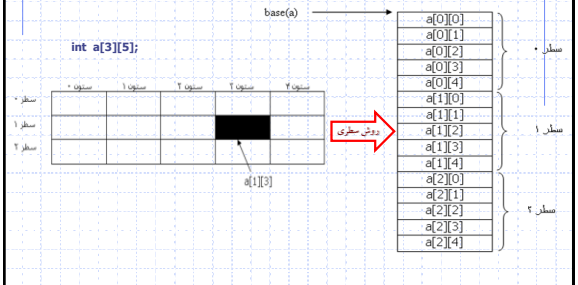
نمایش آرایه ها

- معمولا ارایه چند بعدی از طریق ذخیره سازی عضوهایش در یک آرایه یک بعدی پیاده سازی می شود.
- اگر یک ارایه به صورت $a[upper_0][upper_1]...[upper_n]$ اعلان شده باشد تعداد عضوهای این ارایه برابر است با $\prod_{i=0}^{n-1} upper_i$
- مثال: اگر ما a را به صورت a[10][10][10] اعلان کنیم آنگاه به ۱۰۰۰ واحد حافظه برای ذخیره ان احتیاج داریم
- اگر یک ارایه به صورت $a[p_1..q_1][p_2..q_2]...[p_n..q_n]$ اعلان شده باشد تعداد عضوهای این ارایه برابر است با $\prod_{i=1}^{n}(q_i - p_i + 1)$
- مثال: اگر ما a را به صورت a[3..4] [2..4][1..2][4..5] اعلان کنیم آنگاه این ارایه 2*3*2*2=24 عضو دارد

پیاده سازی آرایه های چند بعدی

- آرایه های دوبعدی می توانند به صورت **سطری** یا **ستونی** ذخیره شوند. در روش سطری، ابتداعناصر سطر اول، سپس عناصر سطردوم و غیره ذخیره می شوند. در روش ستونی، ابتدا عناصر ستون اول، سپس عناصر ستون دوم و غیره ذخیره می شوند.
- در روش سطری آرایه های چند بعدی را به وسیله سطرهای آن ذخیره می کنیم. به عنوان مثال آرایه دو بعدی $A[upper_0][upper_1]$ دارای $upper_0$ سطر به صورت $(row_0, row_1, ..., row_{upper_0-1})$ است به نحوی که هر سطر شامل $upper_1$ عنصر می باشد

پیاده سازی آرایه های چند بعدی



پیاده سازی آرایه های چند بعدی

فرض کنید آرایه A با $upper_0$ سطر و $upper_1$ ستون تعریف شده است و α آدرس A[0][0] باشد،

برای رسیدن به اولین عنصر سطر iأم (یعنی عنصر A[i][0]) باید از i سطر کامل بگذریم که هر سطر آن دارای $upper_1$ عنصر است. لذا آدرس عنصر اول سطر i برابر است با: $\alpha + i. upper_1$ = آدرس اولین عنصر سطر i

فاصله اولین عنصر سطر i تا ستون j برابر با j است. بنابراین آدرس عنصر A[i][j] به صورت زیر است:

آدرس عنصر A[i][j] = $\alpha + i. Upper_1 + j$

پیاده سازی آرایه های چند بعدی

A[upper$_0$][upper$_1$]

- Row major order: A[i][j] : α + i* *upper*$_1$ + j
- Column major order: A[i][j] : α + j* *upper*$_0$ + i

	col$_0$	*col*$_1$	...	*col*$_{u1-1}$
row$_0$	A[0][0] α	A[0][1] α + *u*$_0$	...	A[0][*u*1-1] α+(u$_1$-1)* u$_0$
row$_1$	A[1][0] α + *u*$_1$	A[1][1]	...	A[1][*u*1-1]
	. . .			
row$_{u0-1}$	A[*u*0-1][0] α+(u$_0$-1)*u$_1$	A[*u*0-1][1]	...	A[*u*0-1][*u*1-1]

پیاده سازی آرایه های چند بعدی

برای نمایش آرایه سه بعدی *A*[*upper*$_0$][*upper*$_1$][*upper*$_2$] آن را به عنوان *upper*$_0$ آرایه دو بعدی با ابعاد *upper*$_1$×*upper*$_2$ در نظر می گیریم.

برای پیدا کردن محل a[i][j][k]

address of *a*[*i*][0][0] = α + *i***upper*$_1$**upper*$_2$

چون i آرایه دوبعدی با ابعاد upper1×upper2 قبل از این عضو وجود دارد

address of *a*[*i*][*j*][0]= α + *i***upper*$_1$**upper*$_2$+*j***upper*$_2$

address of *a*[*i*][*j*][*k*]= α + *i***upper*$_1$**upper*$_2$+*j***upper*$_2$+*k*

پیاده سازی آرایه های چند بعدی

- با تعمیم بحث ارائه شده فرمول ادرس هر عضو *A*[*i*$_0$][*i*$_1$]...[*i*$_{n-1}$] در یک آرایه n بعدی که به صورت *A*[*upper*$_0$][*upper*$_1$]...[*upper*$_{n-1}$] به صورت زیر به دست می اید.

where:

$$\begin{cases} a_j = \prod_{k=j+1}^{n-1} upper_k & 0 \le j < n-1 \\ a_{n-1} = 1 \end{cases}$$

$$\begin{aligned} \alpha &+ i_0 upper_1 upper_2 \ldots upper_{n-1} \\ &+ i_1 upper_2 upper_3 \ldots upper_{n-1} \\ &+ i_2 upper_3 upper_4 \ldots upper_{n-1} \\ &\vdots \\ &+ i_{n-2} upper_{n-1} \\ &+ i_{n-1} \end{aligned} = \alpha + \sum_{j=0}^{n-1} i_j a_j$$

نوع داده ای مجرد رشته

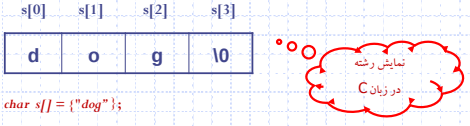
عملکردهایی که می توان برای رشته ها تعریف کرد مانند :

- ایجاد یک رشته تهی جدید
- خواندن یا نوشتن یک رشته
- ضمیمه کردن دو رشته به یکدیگر (*concatenation*)
- کپی کردن یک رشته
- مقایسه رشته ها
- درج کردن یک زیر رشته به داخل رشته
- برداشتن یک زیر رشته از یک رشته مشخص
- پیدا کردن یک الگو(pattern) یا عبارت در یک رشته

نوع داده ای مجرد رشته

نحوه ذخیره سازی در حافظه :

- در زبان C ، رشته ها به صورت آرایه های کاراکتری که به کاراکترتهی، ۰\، ختم می شوند نگهداری می گردد.




```
char s[] = {"dog"};
```

پیدا کردن یک الگو در یک رشته ؟ روش کنوث-موریس-پرات مطالعه شود