مقدمه

ساختمان داده[1] آرایه[2] معمولاً در بیش تر زبان‌های‌برنامه نویسی[3] به صورت از پیش تعریف شده[4] وجود دارد.

مثال: شکل زیر ساختار آرایه ای را نشان می دهد که 8 عنصر دارد.

$$a = \begin{bmatrix} \overbrace{13}^{a[0]}, & \overbrace{-2}^{a[1]}, & \overbrace{0}^{a[2]}, & \overbrace{0}^{a[3]}, & \overbrace{0}^{a[4]}, & \overbrace{0}^{a[5]}, & \overbrace{0}^{a[6]}, & \overbrace{0}^{a[7]} \end{bmatrix}$$

ویژگی بارز آرایه، همگن[5] بودن عناصر[6] اش‌است. یعنی‌طول و اندازه[7] هر کدام از عناصر (بر حسب بایت[8]) با هم برابر هستند. مثلاً همگی‌عدد صحیح[9] (به طول 2 بایت) یا همگی عدد اعشار[10] (به طول 4 بایت) و ...هستند. برای‌دسترسی‌به عناصر، آن‌ها را اندیس‌گذاری[11] می‌کنند. اندیس از 0 شروع می‌شود. اندیس عنصر اول، 0 است. و اندیس‌عنصر دوم، 1 است. و ... . در شکل بالا اندیس هر عنصر نیز نشان داده شده است.

مثال: در زبان سی-دابل-پلاس[12] ساختمان داده ی آرایه، به طور پیش فرض[13] تعریف شده است.

```
int a = {13, -2, 0, 0, 0, 0, 0, 0};
```

ولی جا به جا کردن آن بین توابع راحت نیست. به منظور استفاده بهینه از ساختمان داده آرایه، می توان کتاب خانه[14] *array* را بار گذاری[15] کرد. با این روش روند فرستادن آرایه به توابع[16] تسهیل می شود.

---

1 data structure
2 array
3 programming languages
4 pre-defined
5 homogeneous
6 elements
7 size
8 byte
9 integer
10 float
11 indexing
12 C++
13 default
14 library
15 include
16 pass <array> to function as argument

```
#include <array>
[…]
arr<int,8> = {13, -2, 0, 0, 0, 0, 0, 0};
```

مثال: در زبان پایتون[17]، به طور پیش فرض ساختمان داده لیست[18] موجود ولی ساختمان داده آرایه در دسترس نیست. می‌توان ماژول *numpy* را بارگذاری[19] کرده و از آرایه ها و ویژگی آن‌ها بهره مند شد.

```
Import numpy as np
v = np.array([13, -2, 0, 0, 0, 0, 0, 0])
```

در قطعه کد بالا، *v* آرایه ای با 8 عنصر بوده که همگی از نوع *int* 64 هستند.

در زبان های برنامه نویسی با استفاده از کلاس[20] می‌توان ساختمان داده ها را پیاده سازی کرد. هر کلاس از دو قسمت داده[21] و عملیات روی آن داده ها[22] تشکیل شده است. جدول زیر، نوع داده و عملیات کلاس آرایه را به فرم انتزاعی نمایش می دهد.

	نوع داده انتزاعی آرایه
داده	دنباله[23] ای‌با طول‌ثابت (مجموعه اندیس‌گذاری‌شده) از عناصر که همگن هستند.
عملیات	دستیابی مستقیم به هر عنصر به منظور بازیابی[24] یا ذخیره[25] کردن.

روش های دستیابی به عناصر آرایه

- تصادفی (مستقیم)
- ترتیبی (غیر مستقیم)

دستیابی مستقیم[26] (تصادفی[27]): از آن‌جایی‌که عناصر یک نوع و یک اندازه هستند، آن ها را در بردار پشت سر هم (آدرس های متوالی در حافظه[28]) نگه داری می کنند. به منظور

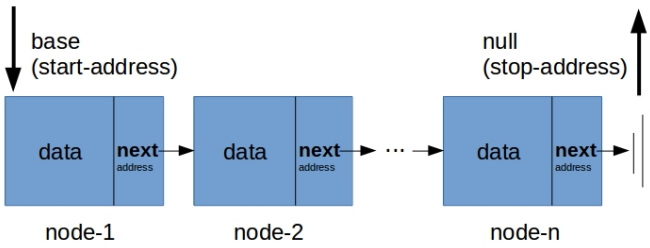
17 Python
18 list
19 import
20 class
21 data member
22 member function
23 sequence
24 retrieve
25 save
26 direct access
27 random
28 memory

دستیابی به هر عنصر از فرمول استفاده می شود. بنابراین زمان دستیابی به هر عنصر، مساوی است با زمان محاسبه فرمول و این محاسبه بسیار سریع انجام می شود.

مثال: زمان دسترسی به عنصر 10 هزارم با زمان دسترسی به عنصر 3 ام، یکی است. زیرا برای پیدا کردن آدرس هر کدام از آن‌ها فقط کافی است از فرمول مربوطه استفاده کنیم.

$$addr\ \ X[3] = base(X) + 3 \times sizeof(data-type)$$
$$addr\ \ X[3] = base(X) + 10{,}000 \times sizeof(data-type)$$

دستیابی ترتیبی[29]: در این‌حالت،همچون‌نوار کاست[30] (که برای رسیدن به رکورد n [31] ام، تمامی رکورد های قبلی باید رد شود.)برای دسترسی‌به عنصر n ام، تمامی عناصر قبل از آن‌باید ملاقات[32] شود. هر عنصر، آدرس عنصر بعدی[33] را نیز در خود نگه داری می کند. به هر کدام از این عناصر که علاوه بر داده، آدرس را نیز نگه داری می کنند، گره [34] گفته می شود.



برای رسیدن به گره n ام، از گره اول شروع می کنیم. از گره اول می پرسیم، آدرس گره دوم چیست. سپس به سراغ گره دوم می رویم. از گره دوم می پرسیم، آدرس گره سوم چیست ... این روند تا رسیدن به گره n ام ادامه دارد.

هر کدام از گره ها (برخلاف عناصر آرایه که پشت سر هم در حافظه قرار داشتند.) در هر محلی از حافظه می توانند قرار بگیرند.علاوه بر این، نوع داده[35] هر کدام از این گره ها می تواند متفاوت باشد. بنابراین طول و اندازه هر کدام از این گره ها متفاوت است. در پیاده

29 Sequential
30 cassette tape
31 track
32 visit
33 next address
34 node
35 data-type

سازی لیست های پیوندی[36]، از این روش (دستیابی ترتیبی[37]) استفاده می شود. پیاده سازی این روش بر مبنای استفاده از اشاره گر ها[38] است.پیدا کردن گره بعدی به این روش در مقایسه با روش مستقیم، بسیار زمان گیر خواهد بود. بنابراین، سرعت اجرایبرنامه پایین می‌آید.

در یک جمع بندی مزایا و معایب ساختمان داده آرایه را بررسی می کنیم.

مزایا: سرعت دسترسی بالا به عناصر که منجر به کاهش زمان اجرا[39] برنامه می شود.

معایب: همه عناصر باید همگن (یک نوع و یک اندازه) باشند.

ابعاد آرایه ها

آرایه ها را می‌توان به 3 دسته:

- تک بعدی (بردار[40])
- دو بعدی (ماتریس[41])
- n بعدی

تقسیم بندی کرد. منظور از بعد[42]، ویژگی[43] عناصر است.مثلاً از ماتریس 3 بعدی $[115,\ 34,\ 40]$

در پردازش تصویر[44] برای مشخص‌کردن مقدار رنگ های (قرمز، سبز، آبی[45]) یک پیکسل[46] که از ویژگی آن نقطه در صفحه نمایش است، استفاده می‌شود.

دسترسی به عناصر آرایه

در ادامه، فرمول دسترسی به عناصر را به ترتیب در بردار، ماتریس و آرایه n بعدی، بررسی می کنیم.

36 linked lists
37 ordered access
38 pointer
39 run-time
40 vector
41 matrix
42 dimension
43 feature
44 image processing
45 (B, G, R) in computer graphic
46 pixel

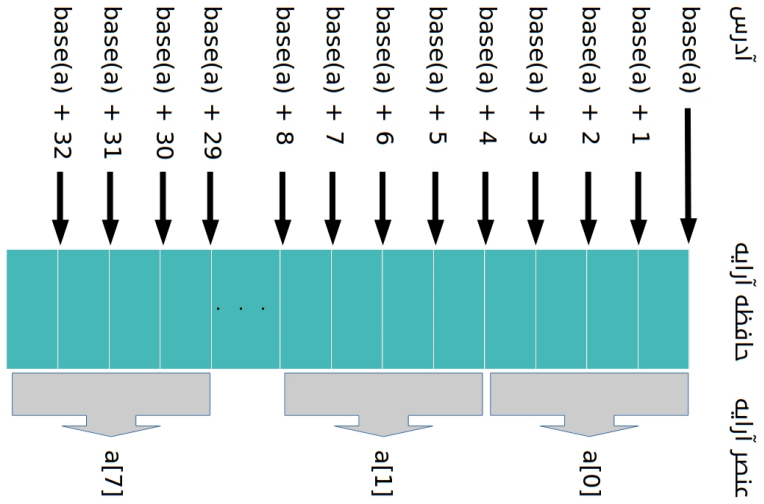
دسترسی به عناصر آرایه تک بعدی (بردار): شکل زیر، یک بردار و طرز قرار گیری چند عنصر و اندیس آن ها (که از صفر شروع می شود.) را نمایش می دهد.

$$a = \begin{bmatrix} \overbrace{v_1}^{a[0]}, & \overbrace{v_2}^{a[1]}, & \overbrace{v_3}^{a[2]}, & \overbrace{v_4}^{a[3]}, & \overbrace{v_5}^{a[4]}, & \overbrace{v_6}^{a[5]}, & \overbrace{v_7}^{a[6]}, & \overbrace{v_8}^{a[7]} \end{bmatrix}$$

به طور کلی، به منظور دسترسی به عناصر یک آرایه تک بعدی، از فرمول زیر استفاده می کنیم.

$$base(a) + i \times sizeof(datatype)$$

از رابطه فوق برای یافتن آدرس عنصر i ام از آرایه a که آدرس اولین عنصر آن (آدرس شروع آرایه)، $base(a)$ است، استفاده می شود. اندازه $size$ عناصر همگن، بر حسب بایت است. شکل زیر روش چیدمان آرایه a را در حافظه نشان می دهد.



مثال: فرض کنید آرایه به صورت

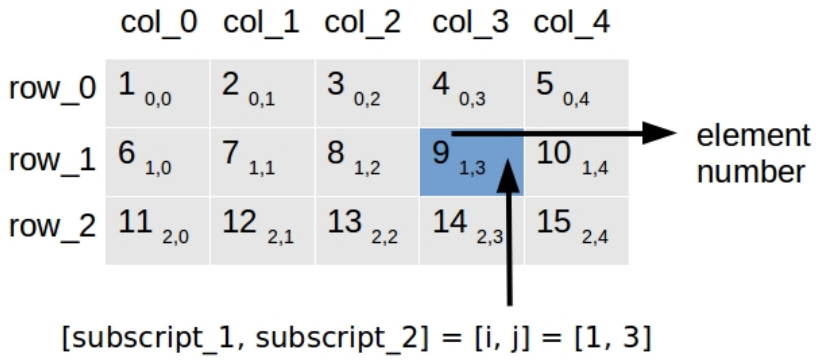
$$integer\ X[10],$$
$$base(X) = 1000,$$
$$sizeof(integer) = 4\ byte$$

تعریف شود. آدرس عنصر $X[3]$ را بیابید.

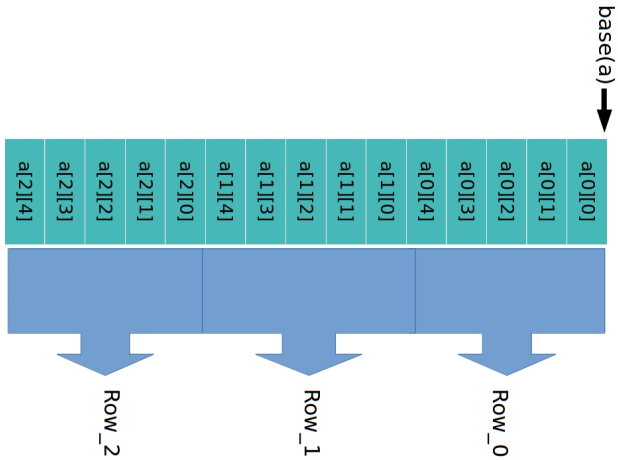
حل:

$$addr\ X[3] =$$
$$base(X) + 3 \times sizeof(integer) =$$
$$1000 + 3 \times 4 = 1012$$

دسترسی به عناصر آرایه 2 بعدی (ماتریس): شکل زیر نمایش هندسی یک آرایه 2 بعدی را نشان می دهد.



شکل زیر نحوه چیدمان عناصر همان آرایه را در حافظه نشان می دهد.



از رابطه زیر می‌توان برای بدست آوردن آدرس عنصری در یک ماتریس، استفاده کرد:

$$addr(a[i][j]) = \{base(a) + i \times n \times sizeof(datatype)\} + \{j \times sizeof(datatype)\} = base(a) + (i \times n + j) \times sizeof(datatype)$$

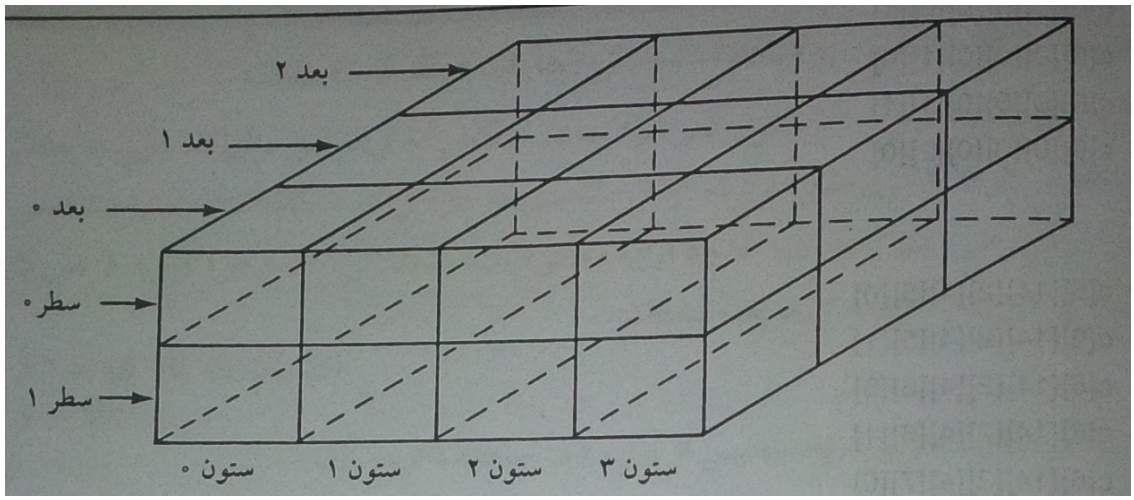
عبارت داخل آکولاد اول، آدرس اولین عنصر سطر i ام می‌باشد و عبارت داخل آکولاد دوم، فاصله اولین عنصر سطر i ام تا ستون j را نشان می دهد.

مثال: آرایه a که m (تعداد سطر[47] اش)و n (تعداد ستون[48] اش)به ترتیب 3 و 5 می باشد، را در نظر بگیرید. آدرس عنصر $a[1][3]$ را بیابید.

حل:

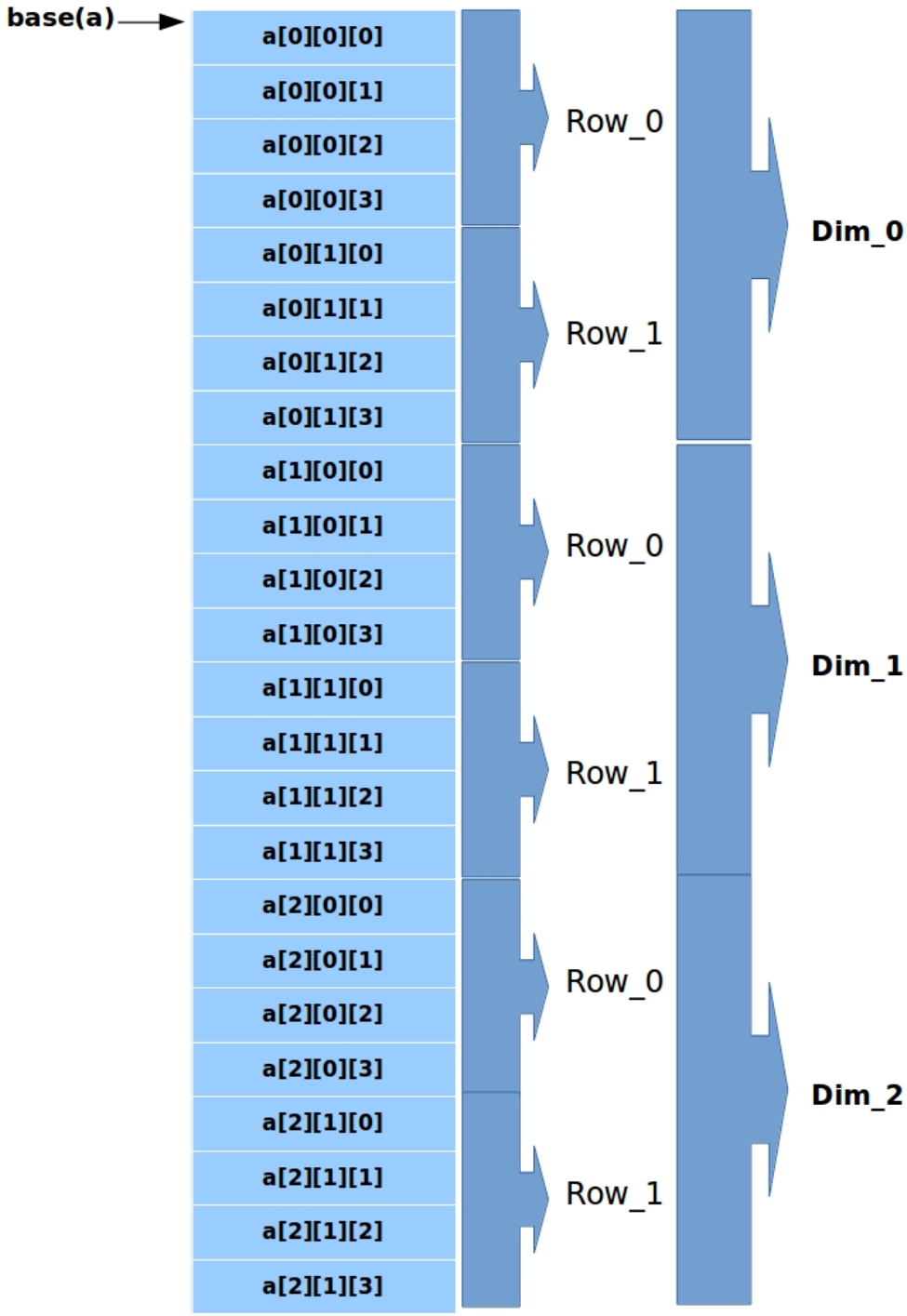
$$addr(a[1][3]) = base(a) + (1 \times 5 + 3) \times 4 = base(a) + 32$$

به هر حال، می توان با نرم افزار هایی همچون متلب[49]، شکلی هندسی از آرایه های چندین بعدی ترسیم کرد. ولی نمایش آن ها به صورت هندسی که خیلی هم پیچیده می شود، صرفاً برای به رخ کشیدن قابلیت های این جور نرم افزار ها است و به فهم و درک ارتباط داده ها کمک زیادی نمی کند. شکل زیر یک تجسم هندسی از آرایه3 بعدی را نمایش می دهد.



به منظور بهتر متوجه شدن چید مان عناصر در حافظه، می توان آرایه n بعدی (در این مثال 3 بعدی) را به فرم خطی (سطری) تبدیل کرد و پشت سر هم بنویسیم.

47 row
48 column
49 MATLAB


base(a)
a[0][0][0]
a[0][0][1]
a[0][0][2]
a[0][0][3]
a[0][1][0]
a[0][1][1]
a[0][1][2]
a[0][1][3]
a[1][0][0]
a[1][0][1]
a[1][0][2]
a[1][0][3]
a[1][1][0]
a[1][1][1]
a[1][1][2]
a[1][1][3]
a[2][0][0]
a[2][0][1]
a[2][0][2]
a[2][0][3]
a[2][1][0]
a[2][1][1]
a[2][1][2]
a[2][1][3]
Row_0
Row_1
Row_0
Row_1
Row_0
Row_1
Dim_0
Dim_1
Dim_2

فرمول دسترسی به عناصر آرایه n بعدی: تلاش می‌کنیم با بررسی کردن چند بعد بالاتر و مطالعه روابط آن ها (به این روش استقرای ریاضی[50] می گویند.)،فرمولی برای دسترسی به عناصر آرایه $n-dim$ پیدا کنیم. متوجه می‌شویم که اندیس‌خارجی‌ها سرعت بیشتری تغییر می‌کند. همانند کیلومتر شمار ماشین‌که رقم سمت راست‌آن‌با سرعت بیشتر ی تغییر می کند.

آرایه n بعدی a را در نظر بگیرید.

$$integer\, a[r_1][r_2]...[r_n]$$

فرض کنید آرایه به صورت خطی[51] (سطری) تبدیل و نمایش داده شود. طول هرعنصر را $size$ و آدرس‌اولین‌عنصر را $base(a)$ در نظر بگیرید. برای دسترسی‌به عنصر $a[i_1][i_2]...[i_n]$

از فرمول زیر استفاده می کنیم:

$$base(a) + size \times [i_1 \times (r_2 \times ... \times r_n) + i_2 \times (r_3 \times ... \times r_n) + ... + i_{(n-1)} \times (r_n) + i_n]$$

برنامه زیر برای محاسبه فرمول بالا می تواند کارآمد باشد:

```
offset = 0;
for(int j=0; j<n; j++)
   offset = r[j] * offset + i[j];
addr = base(a) + size * offset
```

در قطعه کد بالا i و r آرایه هایی به طول n که به ترتیب اندیس و بازه ی آن اندیس می باشند.

کاربرد های آرایه

از مهم ترین کاربرد های آرایه، در مرتب سازی[52] و جست‌و جو [53] می توان‌یاد کرد. علاوه بر آن، این‌ساختمان داده، پایه و مبنای ایجاد بقیه ساختمان داده ها نیز هست‌به عنوان مثال، یکی از روش های پیاده سازی ساختمان‌داده درخت[54]، استفاده از آرایه است توضیح دقیق‌تر الگوریتم های‌جست‌و جو و محاسبه پیچیدگی زمانی[55] آن ها، از حوصله این بحث خارج است. بنابراین‌در این‌قسمت صرفا به ذکر عناوین‌می‌پردازیم:

50 inductive reasoning
51 flatten
52 sort
53 search
54 tree
55 time complexity (O_n)

- الگوریتم انتخابی[56]
- الگوریتم خطی[57]
- الگوریتم دودویی[58]
- ...

عملیات روی آرایه ها

با عنایت به این مهم که فرم های شکل دهی مختلفی از آرایه (تک بعدی، دو بعدی، چند بعدی) وجود دارد، عملیات[59] های متفاوتی می‌توان روی آن ها انجام داد. بسیاری از این عملیات متداول بوده و در کتابخانه[60] ها و ماژول[61] های زبان های برنامه نویسی[62] وجود دارد. بنابراین در این جا به ذکر عناوین اکتفا می شود:

- جمع و تفریق ماتریس ها
- ضرب داخلی[63] و خارجی[64] ماتریس‌ها
- ترانهاده[65] ماتریس‌ها
- ماتریس های بالا مثلثی[66] و پایین مثلثی[67]
- ماتریس اسپارس
- ترانهاده ماتریس های اسپارس
- جمع و تفریق ماتریس های اسپارس
- ...

از آن جایی که استفاده از ماتریس اسپارس تاثیر بسزایی در ذخیره کردن داده ها و صرفه جویی در مصرف حافظه[68] دارد، نقش بسیار مهم و کاربردی در مباحث مربوط به علم

56 selection sort
57 linear search
58 binary search
59 operation
60 library
61 module
62 programming language
63 inner product
64 outer product
65 transpose
66 upper triangular
67 lower triangular
68 memory efficiency

داده[69] و داده کاوی[70] ایفا می‌کند. بنابراین در این‌بخش به توضیح ماتریس‌اسپارس (ماتریس خلوت) می‌پردازیم.

ماتریس اسپارس (خلوت): به ماتریسی گویند که تعداد زیادی از عناصر آن صفر[71] باشد. عملیاتی که روی یک ماتریس انجام می شود، روی عناصر صفر آن اجرا نمی شود. ممکن است تعداد عناصر صفر یک ماتریس خیلی زیاد باشد. به چنین ماتریسی با عناصر صفر زیاد[72]، ماتریس خلوت (اسپارس) می گویند. نگه داری چنین ماتریسی، در حافظه به صرفه نیست. بنابراین فرمت دیگری برای نگه داری آن ها نیاز است که در مصرف حافظه صرفه جویی شود.

مثال: یک نمونه ماتریس اسپارس:

$$A = \begin{bmatrix} 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 4 \\ 0 & 0 & 0 & 5 & 0 & 0 \end{bmatrix}$$

به منظور ارتقا کیفیت نمایش، عناصر صفر را با نقطه جایگذاری می کنیم. بدین ترتیب فقط عناصر غیر صفر که تعداد آن ها معدود است، به چشم می آید:

$$A = \begin{bmatrix} . & . & . & 5 & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & 5 & . & 4 \\ . & . & . & 5 & . & . \end{bmatrix}$$

مثال: نمایش ماتریس اسپارس به فرمت فشرده:

$$AS = \begin{bmatrix} R=5 & C=6 & NZ=2 \\ 0 & 2 & 5 \\ 3 & 0 & 4 \end{bmatrix}$$

$R: \ number \ \ of \ \ rows$
$C: \ number \ \ of \ columns$
$NZ: \ number \ \ of \ \ (N)on\text{-}(Z)ero \ \ elements$

69 data science
70 data mining
71 zero
72 dense

مثال: مقایسه حافظه مصرفی فرمت معمولی (A) و فرمت فشرده[73] (AS) :

$$Space(A) = 5 \times 6 \times (sizeof(integer): 2\ byte) = 60\ byte$$
$$Space(AS) = 3 \times 3 \times (sizeof(integer): 2\ byte) = 18\ byte$$

مراجع

۱. ساختمان داده ها در C++، عین الله جعفر نژاد قمی، 2003

2. Data Structures Using C and C++ (2nd Edition), Yedidyah Langsam, 1994

73 compressed form