مقدمه

ساختمان داده[1] آرایه[2] معمولاً در بیش تر زبان های برنامه نویسی[3] به صورت از پیش تعریف شده[4]، وجود دارد. ویژگی بارز آرایه، همگن[5] بودن عناصر[6] اش است. یعنی طول و اندازه[7] هر کدام از عناصر (بر حسب بایت[8]) با هم برابر هستند. مثلاً همگی عدد صحیح[9] (به طول 2 بایت) یا همگی عدد اعشار[10] (به طول 4 بایت) و ... هستند. برای دسترسی به عناصر، آن ها را اندیس گذاری[11] می کنند. اندیس از 0 شروع می شود. اندیس عنصر اول، 0 است. و اندیس عنصر دوم، 1 است. و ...

نوع داده انتزاعی آرایه

داده: دنباله[12] ای با طول ثابت (مجموعه اندیس گذاری شده) از عناصر که همگن هستند.
عملیات: دستیابی مستقیم به هر عنصر به منظور بازیابی[13] یا ذخیره[14] کردن.

دستیابی مستقیم[15] (تصادفی[16])

از آن جایی که عناصر یک نوع و یک اندازه هستند، آن ها را در بردار پشت سر هم (آدرس های متوالی در حافظه[17]) نگه داری می کنند. به منظور دستیابی به هر عنصر از فرمول استفاده می شود. بنابراین زمان دستیابی به هر عنصر مساوی است با زمان محاسبه فرمول. به عنوان مثال، زمان دسترسی به عنصر 10 هزارم، برابر است با زمان دسترسی به عنصر 3 ام.

---

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 sequence
13 retrieve
14 save
15 direct access
16 random
17 memory

دستیابی ترتیبی[18]: در این حالت، همچون نوار کاست[19] (که برای رسیدن به رکورد[20] n ام، تمامی رکورد های قبلی باید رد شود.) برای دسترسی به عنصر n ام، تمامی عناصر قبل از آن باید ملاقات[21] شود. هر عنصر، آدرس عنصر بعدی[22] را نیز در خود نگه داری می کند. به هر کدام از این عناصر که علاوه بر داده، آدرس عنصر بعدی را نیز نگه داری می کنند، گره[23] 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 گره‌ها (برخلاف عناصر آرایه که پشت سر هم در حافظه قرار می گیرند.) در هر محلی از حافظه می‌توانند قرار داشته باشند. نوع داده[24] هر کدام از این گره ها می‌تواند متفاوت باشد. بنابراین طول و اندازه هر کدام از این گره ها متفاوت است. در پیاده‌سازی لیست های پیوندی[25]، از این روش (دستیابی ترتیبی[26]) استفاده می شود. پیدا کردن گره بعدی به این روش در مقایسه با روش مستقیم، بسیار زمان بر خواهد بود. بنابراین، سرعت اجرای برنامه پایین می آید.

مزیت آرایه: سرعت دسترسی بالا به عناصر که منجر به کاهش زمان اجرا[27] برنامه می شود.

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

1 Array Dimensions (Shapes)

آرایه چندین بعدی: آرایه ها را به 3 دسته:

- تک بعدی (بردار[28])
- دو بعدی (ماتریس[29])
- n بعدی

18 Sequential
19 Cassette Tape
20 Track
21 Visit
22 Next Address
23 Node
24 Data-Type
25 Linked List
26 Ordered Access
27 Run-Time
28 Vector
29 Matrix

می توان تقسیم بندی کرد. منظور از بعد[30]، ویژگی[31] عناصر است. مثلاً ماتریس 3 بعدی $(115,\ 34,\ 40)$ در پردازش تصویر[32] برای مشخص کردن رنگ های (قرمز، سبز، آبی[33]) یک پیکسل[34] که از ویژگی آن نقطه در صفحه نمایش است، استفاده می شود. به هر حال، می توان با نرم افزار هایی همچون متلب[35]، شکلی هندسی از آرایه های چندین بعدی ترسیم کرد. ولی نمایش آن‌ها به صورت هندسی که خیلی هم پیچیده می شود، صرفاً برای به رخ کشیدن قابلیت های این جور نرم افزار هاست. و به فهم و درک ارتباط داده‌ها کمک زیادی نمی کند.

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

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

به طور کلی، به منظور دسترسی به عناصر یک آرایه تک بعدی، ازفرمول 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 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:

30 Dimension
31 Feature
32 Image Processing
33 (B,G,R) for computer graphic
34 Pixel
35 MATLAB

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

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

شکل نحوه چیدمان عناصر یک آرایه 2 بعدی (ماتریس) را درحافظه 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[36]) و n (تعداد ستون[37]) به ترتیب 3 و 5 می باشد، را در نظر بگیرید. آدرس عنصر $a[1][3]$ را بیابید.

حل:

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

می توان با نرم افزار هایی همچون متلب، شکلی هندسی از آرایه های چند بعدی ترسیم کرد. ولی نمایش آن‌ها به صورت هندسی که خیلی هم پیچیده می شود، صرفاً برای به رخ کشیدن قابلیت های این جور نرم افزار هاست. و به فهم و درک ارتباط داده‌ها کمک زیادی نمی کند. به منظور بهتر متوجه شدن چیدمان عناصر در حافظه، می توان آرایه n بعدی (در این مثال 2 بعدی) را به فرم خطی (سطری) تبدیل کرد و پشت سر هم بنویسیم.

فرمول دسترسی به عناصر آرایه n بعدی: با بررسی کردن چند بعد دیگر و مطالعه روا بط آن‌ها و همچنین از طریق استقرای ریاضی[38] متوجه می شویم که اندیس خارجی با سرعت بیشتری تغییر می کند. همانند کیلومتر شمار ماشین که رقم سمت راست آن با سرعت بیشتری تغییر می کند. آرایه n بعدی a را در نظر بگیرید.

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

36 Row
37 Column
38 Inductive Reasoning

فرض کنید آرایه به صورت خطی[39] (سطری) تبدیل و نمایش داده شود. طول هر عنصر را $size$ و آدرس اولین عنصر را $base(a)$ در نظر بگیرید. برای دسترسی به عنصر $a[i_1][i_2]...[i_n]$ ، از فرمول زیر استفاده می کنیم:

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 آن اندیس می باشند.

2 Array Applications

کاربرد های آرایه: از مهم ترین کاربرد های آرایه، در مرتب سازی[40] و جست و جو[41] می توان یاد کرد. علاوه بر آن ، ا ین ساختمان داده، پایه و بنای بقیه ساختمان داده ها نیز هست. به عنوان مثال، یکی از روش های پیاده‌سازی ساختمان داده درخت، استفاده از آرایه است. توضیح دقیق تر الگوریتم های جست و جو و محاسبه پیچیدگی زمانی[42] آن ها، از حوصله این بحث خارج است. بنابراین در این قسمت صرفا به ذکر عناوین می پردازیم:

- الگوریتم انتخابی[43]
- الگوریتم خطی[44]
- الگوریتم دودویی[45]
- ...

39 Flatten
40 Sort
41 Search
42 Time Complexity (O_n)
43 Selection Sort Algorithm
44 Linear Search Algorithm
45 Binary Search Algorithm

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[46] های متفاوتی را می توان روی آن ها انجام داد. بسیاری از این عملیات متدوال بوده ودر کتابخانه[47] ها و ماژول[48] های زبان های بر نامه نویسی[49] وجود دارد. بنابراین دراین جا به ذکر عناوین اکتفا می شود:

- جمع و تفریق ماتریس ها
- ضرب داخلی[50] و خارجی[51] ماتریس ها
- ترانهاده[52] ماتریس ها
- ماتریس های بالا مثلثی[53] و پایین مثلثی[54]
- ماتریس اسپارس
- ترانهاده ماتریس های اسپارس
- جمع و تفریق ماتریس های اسپارس
- ...

ازآن جایی که استفاده از ماتریس اسپارس تاثیر بسزایی در ذخیره کردن داده‌ها و صرفه جویی در مصرف حافظه[55] دارد، نقش بسیار مهم وکاربردی در مباحث مربوط به علم داده[56] و داده کاوی[57] ایفا می کند. بنا براین دراین بخش به توضیح ماتریس اسپارس (ماتریس خلوت) می پردازیم.

Sparse Matrix

ماتریس اسپارس (خلوت): به ماتریسی گویند که تعداد زیادی از عناصر آن صفر[58] باشد.

46 Operation
47 Library
48 Module
49 Programming Language
50 Inner Product
51 Outer Product
52 Transpose
53 Upper Triangular
54 Lower Triangular
55 Memory Efficiency
56 Data Science
57 Data Mining
58 Zero

عملیاتی که روی یک ماتریس انجام می شود، روی عناصر صفر آن اجرا نمی شود. ممکن است تعداد عناصر یک ماتریس خیلی زیاد باشد. به چنین ماتریسی با عناصر صفر زیاد[59]، 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 آن‌ها معدود است، به چشم می آید:

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

مثال: نمایش ماتریس اسپارس به فرمت فشرده:

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

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

مثال: مقایسه حافظه مصرفی فرمت معمولی (A) و فرمت فشرده[60] (AS) :

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

59 Dense
60 Compressed Form