عنوان :

الگوریتم های مرتب سازی

استاد ارجمند :

جناب آقای مهندس فرزام پور

مقدمه :

الگوریتم مرتب‌سازی، در علوم کامپیوتر و ریاضی، الگوریتمی است که لیستی از داده‌ها را به ترتیبی مشخص می‌چیند.

پر استفاده‌ترین ترتیب‌ها، ترتیب‌های عددی و لغت‌نامه‌ای هستند. مرتب‌سازی کارا در بهینه سازی الگوریم‌هایی که به لیست‌های مرتب شده نیاز دارند (مثل جستجو و ترکیب) اهمیت زیادی دارد.

از ابتدای علم کامپیوتر مسائل مرتب‌سازی تحقیقات فراوانی را متوجه خود ساختند، شاید به این علت که در عین ساده بودن، حل آن به صورت کارا پیچیده‌است. برای مثال مرتب‌سازی حبابی در سال ۱۹۵۶ به وجود آمد. در حالی که بسیاری این را یک مسئلهٔ حل شده می‌پندارند، الگوریتم کارآمد جدیدی همچنان ابداع می‌شوند (مثلاً مرتب‌سازی کتاب خانه‌ای در سال ۲۰۰۴ مطرح شد).

مبحث مرتب‌سازی در کلاس‌های معرفی علم کامپیوتر بسیار پر کاربرد است، مبحثی که در آن وجود الگوریتم‌های فراوان به آشنایی با ایده‌های کلی و مراحل طراحی الگوریتم‌های مختلف کمک می‌کند؛ مانند تحلیل الگوریتم، داده‌ساختارها، الگوریتم‌های تصادفی، تحلیل بدترین و بهترین حالت و حالت میانگین، هزینهٔ زمان و حافظه، و حد پایین.

در علم کامپیوتر معمولاً الگوریتم‌های مرتب‌سازی بر اساس این معیارها طبقه‌بندی می‌شوند:

- پیچیدگی (بدترین و بهترین عملکرد و عملکرد میانگین): با توجه به اندازهٔ لیست (n). در مرتب‌سازی‌های معمولی عملکرد خوب (O(n log n و عملکرد بد (O(n²) است. بهترین عملکرد برای مرتب‌سازی (O(n است. الگوریتم‌هایی که فقط از مقایسهٔ کلیدها استفاده می‌کنند در حالت میانگین حداقل (O(n log n مقایسه نیاز دارند.
- حافظه (و سایر منابع کامپیوتر) : بعضی از الگوریتم‌های مرتب‌سازی «در جا[1]» هستند. یعنی به جز داده‌هایی که باید مرتب شوند، حافظهٔ کمی (O(۱)) مورد نیاز است؛ در حالی که سایر الگوریتم‌ها به ایجاد مکان‌های کمکی در حافظه برای نگه‌داری اطلاعات موقت نیاز دارند.
- پایداری[2] : الگوریتم‌های مرتب‌سازی پایدار ترتیب را بین داده‌های دارای کلیدهای برابر حفظ می‌کنند. فرض کنید می‌خواهیم چند نفر را بر اساس سن با یک الگوریتم پایدار مرتب کنیم. اگر دو نفر با نام‌های الف و ب هم‌سن باشند و در لیست اولیه الف جلوتر از ب آمده باشد، در لیست مرتب شده هم الف جلوتر از ب است.
- مقایسه‌ای بودن یا نبودن. در یک مرتب‌سازی مقایسه‌ای داده‌ها فقط با مقایسه به وسیلهٔ یک عملگر مقایسه مرتب می‌شوند.
- روش کلی : درجی، جابجایی، گزینشی، ترکیبی و غیره. جابجایی مانند مرتب‌سازی حبابی و مرتب‌سازی سریع و گزینشی مانند مرتب‌سازی پشته‌ای.

الگوریتم‌های مرتب سازی

[ویرایش] مرتب سازی حبابی

(به انگلیسی: Bubble Sort)

فرض کنید n داده داریم که می‌خواهیم به صورت صعودی مرتب شوند. عنصر اول رو با دومی مقایسه ، و در صورتی که اولی بزرگتر باشد جاهاشون رو عوض می کنیم. همین کار رو با عناصر دوم و سوم انجام می‌دهید و همینطور عناصر سوم و چهارم ، الی آخر. وقتی این کار تموم شد بزرگترین عنصر بین داده‌ها به آخر لیست می‌رسد . حالا یک بار دیگه از اول این کار رو انجام می‌دهیم اما این بار تا عنصر (n - ۱)ام ادامه می‌دهیم (عنصر nام مرحله اول جای خودش رو پیدا کرده). باز هم این کار رو تا عنصر (n - ۲)ام تکرار می کنیم ، و بازهم .... تا اینکه بالاخره داده‌ها مرتب می‌شوند. مثلا:

۰ – ۰) ۵ ۶ ۴ ۲

۱ – ۱) ۵ ۶ ۴ ۲

۱ – ۲) ۵ ۴ ۶ ۲

۱ – ۳) ۵ ۴ ۲ ۶

۲ – ۱) ۴ ۵ ۲ ۶

۲ – ۲) ۴ ۲ ۵ ۶

۳ – ۱) ۲ ۴ ۵ ۶

مرحله اول سه مقایسه ، مرحله دوم دو مقایسه و مرحله سوم یک مقایسه داره ، که روی هم می‌شوند شش مقایسه. در کل این روش $n(n-1)/2$ مقایسه لازم داره. اما نه همیشه. به مثال زیر توجه کنید:

۰ – ۰) ۰ ۷ ۱ ۳ ۵ ۴

۱ – ۱) ۰ ۱ ۷ ۳ ۵ ۴

۱ – ۲) ۰ ۱ ۷ ۳ ۵ ۴

۱ – ۳) ۰ ۱ ۳ ۷ ۵ ۴

۱ – ۴) ۰ ۱ ۳ ۵ ۷ ۴

۱ – ۵) ۰ ۱ ۳ ۵ ۴ ۷

۲ – ۱) ۰ ۱ ۳ ۵ ۴ ۷

۲ – ۲) ۰ ۱ ۳ ۵ ۴ ۷

۲ – ۳) ۰ ۱ ۳ ۵ ۴ ۷

۲ – ۴) ۰ ۱ ۳ ۴ ۵ ۷

۳ – ۱) ۰ ۱ ۳ ۴ ۵ ۷

۳ – ۲) ۰ ۱ ۳ ۴ ۵ ۷

۳ – ۳) ۰ ۱ ۳ ۴ ۵ ۷

۴ – ۱) ۰ ۱ ۳ ۴ ۵ ۷

۴ – ۲) ۰ ۱ ۳ ۴ ۵ ۷

۵ – ۱) ۰ ۱ ۳ ۴ ۵ ۷

همونطور که می‌بینید انتهای مرحله ۲ داده‌ها مرتب هستن. تشخیص این مساله هم کار سختی نیست: اگه به مرحله‌ای رسیدیم که هیچ جابجایی در اون رخ نداد نتیجه می‌شه که داده‌ها مرتب

هستن (مرحله سوم). پس بعد از مرحله ۳ مطمئن می‌شیم که داده هامون مرتب شدن و نیازی به مراحل ۴ و ۵ نیست. پیاده سازی (مرتب سازی حبابی) در C++

```
void bubble_sort (int arr [ ] , int n)

{

  register int i , j , t , c;

(--  for (i = n - ۲ ; i >= ۰ ; i

  {

    c = ۰;

   (++ for (j = ۰ ; j <= i ; j

       if (arr [ j ] > arr [ j + ۱ ])

    {

    ; ] t = arr [ j
```

```
            arr [ j ] = arr [ j + ١ ];


        ; arr [ j + ١ ] = t

         C++;

    }

      (if (c == ٠

        ; break

   }
}
```

مرتب سازی سریع

مرتب سازی سریع(به انگلیسی: Quicksort)، یکی از روش های مرتب سازی آرایه است که به دلیل مصرف حافظه کم، سرعت اجرای مناسب و پیاده سازی ساده بسیار مورد قبول واقع شده است.

فهرست مندرجات

[مخفی شود]

پیاده سازی

هر پیاده سازی این الگوریتم به صورت کلی از دو بخش تشکیل شده است. یک بخش تقسیم بندی آرایه (partition) و قسمت مرتب کردن.

پیاده سازی به زبان سی پلاس پلاس

نمونه ای از این پیاده سازی به زبان C++ به صورت زیر است

```
void quicksort(int array[] , int left , int right){

if (left < right){

int middle = partition(array , left , right) ;
quicksort(array , left , middle-۱) ;

quicksort(array , middle+۱ , right);

}
}
int partition(int array[] , int left , int right){
int middle ;
int x = array[left] ;
int l = left ;
int r = right ;
while(l < r){
```

```
while((array[l] <= x) && (l < right)) l++ ;
while((array[r] > x) && (r >= left)) r-- ;
if(l < r){
int temp = array[l];
array[l] = array[r];
array[r] = temp ;
}
}
middle = r ;
int temp = array[left];
array[left] = array[middle] ;
array[middle] = temp; return middle ; }
```

پیاده سازی به زبان پاسکال

پیاده سازی مشابه ولی فشرده تر به زبان pascal به صورت زیر می تواند باشد

```
procedure Sort(l, r: Integer);

var i, j, x, y: integer;
begin
i := l; j := r; x := a[(l+r) DIV ۲];
repeat
```

```
while a[i] < x do i := i + ۱;

while x < a[j] do j := j - ۱;

if i <= j then
begin
y := a[i]; a[i] := a[j]; a[j] := y;
i := i + ۱; j := j - ۱;

end;
until i > j;
if l < j then Sort(l, j);
if i < r then Sort(i, r);
end;
```

{

پیاده سازی به صورت تصادفی

در پیاده سازی الگوریتم quickSort به صورت Randomized تغییر اصلی در بخش Partition کردن آرایه رخ می دهد مه ما درایه a[i] x را به صورت تصادفی انتخاب می کنیم. تنها نکته مهم دراین زمینه این است که در این پیاده سازی باید دقت شود که انتخاب درایه جدید به قبلی ها وابسته نشود و هر بار یک درایه جدید را اختیار نماید.

پیاده سازی صنعتی

الگوریتم مرتب سازی در دنیای واقعی برای آرایه نسبتا کوچک مناسب نیست. به علاوه بخش پارتیشن خود نیز مشکل بزرگی در زمان اجرا می باشد. برای همین پیشنهاد می گردد برای آرایه هایی از طول کمتر از ۷ از مرتب سازی های دیگر مانند مرتب سازی درجی یا حبابی استفاده شود. به علاوه بجای پیاده سازی بخش partition به صورت عادی با احتمالاتی می توان از میانه ۹ برای آرایه های بزرگ (بیش از ۴۰ درایه) و میانه ۳ برای ارایه های متوسط (کمتر از ۴۰ درایه) و عضو وسط برای آرایه های کوچک استفاده کرد. به علاوه در چنین پیاده سازی هایی ابتدا اعداد صفر (برای آرایه از اعداد مثبت) را ابتدا به شروع آرایه منتقل می کنند. و همچنین درایه های غیر عددی را نیز هندل می کنند تا در اجرای الگوریتم اختلالی به وجود نیاورد.

برای توضیحات بیشتر درباره نسخه های بهینه مرتب سازی سریع می توانید به مرجع بنتلی و مک ایلوری مراجعه نمایید. پیاده سازی بسیار خوبی از این الگوریتم را می توانید در کد منبع جاوا و در کلاس java.util.Array بیابید.

زمان اجرا

مرتب سازی سریع چه در پیاده سازی عادی و چه در پیاده سازی احتمالی در حالت متوسط در زمان اجرای O(n lgn) اجرا می شود. چرا که رابطه بازگشتی T(n) = ۲T(n/۲) + Theta(n) x درباره آن صدق میکند. در بدترین حالت زمان اجرای این الگوریتم Theta(n^۲)\ است.

مراجع

الگوریتم QuickSort را می توان در هر مرجع داده ساختار و الگوریتم یافت. با این وجود مراجع زیر (به خصوص کتاب کناث) برای مطالعات بعدی توصیه می شود.

- D.E.Knuth «The Art of Computer Programming» Vol.۲
- Udi Manber , Introduction to Algorithms- A creative Approach
- CLRS , Introduction to Algorithms
- Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function