مقایسهٔ الگوریتم‌های مرتب‌سازی

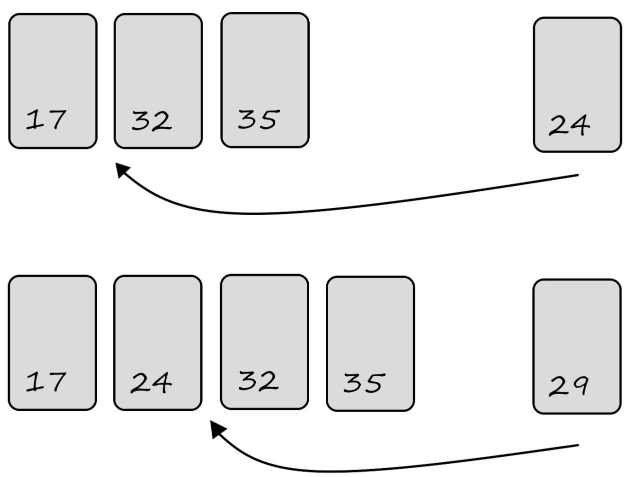
محمد ترابی - علی جعفرآبادی - رضا تاج‌گذاری

تیرماه ۱۴۰۲

۱ مرتب‌سازی درجی[1]

اگر یک دسته کارت به شما داده شود که اعداد ۱ تا ۵۰ روی آن نوشته شده است، چگونه آن را مرتب[2] می‌کنید؟ احتمالا اول تعداد کمی کارت برمی‌دارید و آن‌ها را مرتب می‌کنید؛ سپس بقیه کارت‌ها را یکی پس از دیگری نگاه می‌کنید و در جای مناسب میان کارت‌های مرتب شده قرار می‌دهید. شکل ۱ نمایی کلی از این روش مرتب‌سازی نشان می‌دهد.

وقتی کارت‌ها را با این روند مرتب می‌کنیم، همواره تعدادی از کارت‌ها مرتب شده است و کارت‌هایی که هنوز مرتب نشده، یکی پس از دیگری در دستهٔ کارت‌های مرتب شده درج[3] می‌شوند. اگر با این روش کارت‌ها را مرتب کنیم، درواقع از مرتب‌سازی درجی استفاده کرده‌ایم.



شکل ۱: مراحلی از مرتب کردن کارت‌ها به کمک مرتب‌سازی درجی

[1] Insertion sort
[2] sort
[3] insert

الگوریتم مرتب‌سازی درجی

آرایه‌ای به طول یک همواره مرتب است؛ بنابراین از عنصر دوم شروع می‌کنیم و جلو می‌رویم. فرض کنیم که به عنصر i رسیده‌ایم، عناصر قبل از i مرتب‌اند. لذا از آخرین عنصر قبل از i شروع می‌کنیم و به عقب می‌رویم. هر یک از عناصر بزرگتر از i را یک واحد به سمت راست انتقال[۴] می‌دهیم. وقتی به عنصری کوچک تر از i یا به ابتدای آرایه رسیدیم متوقف می‌شویم و i را همانجا درج می‌کنیم. هنگامی که عنصر i را درج می‌کنیم، i عنصر اول آرایه مرتب می‌شوند. بنابراین این کار را ادامه می‌دهیم تا تمام عناصر آرایه مرتب شوند. درستی مرتب‌سازی درجی را می‌توان به کمک ثابت‌های حلقه[۵] اثبات کرد. [۱] شبه‌کد مرتب‌سازی درجی را می‌توانید در الگوریتم ۱ که در ادامه آمده است مشاهده کنید [۶] .

الگوریتم ۱ مرتب‌سازی درجی

procedure InsertionSort(arr, n)
    for $i \leftarrow 1$ **to** $n-1$ **do**
        $key \leftarrow arr[i]$
        $j \leftarrow i-1$
        while $j >= 0$ and $arr[j] > key$ **do**
            $arr[j+1] \leftarrow arr[j]$
            $j \leftarrow j-1$
        $arr[j+1] \leftarrow key$

مثال ۱ *اگر الگوریتم بالا را روی آرایهٔ [۷, ۶, ۶, ۴, ۹] اجرا کنیم، آرایه بدین صورت مرتب می‌شود:*

$$[۷, ۶, ۶, ۴, ۹] \rightarrow [۷, ۷, ۶, ۴, ۹] \rightarrow [۶, ۷, ۶, ۴, ۹] \rightarrow [۶, ۷, ۷, ۴, ۹] \rightarrow$$
$$[۶, ۶, ۷, ۴, ۹] \rightarrow [۶, ۶, ۷, ۷, ۹] \rightarrow [۶, ۶, ۶, ۷, ۹] \rightarrow [۶, ۶, ۶, ۷, ۹] \rightarrow$$
$$[۴, ۶, ۶, ۷, ۹] \rightarrow [۴, ۶, ۶, ۷, ۹]$$

[4] shift
[5] loop invariants
[۶] توجه داشته باشید که الگوریتم بر پایه صفر است؛ یعنی عنصر اول آرایه $arr[۰]$ می‌باشد.

توجه داشته باشید که پیچیدگی زمانی مرتب‌سازی درجی در بهترین حالت خطی است. همچنین بهترین حالت وقتی اتفاق می‌افتد که آرایه مرتب باشد. می‌توانیم نتیجه بگیریم که در مواقعی که آرایه مرتب و یا تقریبا مرتب است مرتب‌سازی درجی بسیار سریع عمل می‌کند. بنابراین اگر از قبل مطلع هستیم که معمولا داده‌های ما تقریبا مرتب است، استفاده از مرتب‌سازی درجی می‌تواند گزینه مناسبی باشد.

به عنوان مثال فرض کنید که یک لیست هزارتایی از اسامی دانشجویان یک موسسه در اختیار دارید که به ترتیب حروف الفبا مرتب شده اند. در سال جدید پنجاه دانشجو در موسسه نام نویسی می‌کنند و اسامی آن‌ها به آخر آرایه اضافه می‌شود. اگرچه روش‌های زیادی برای مرتب کردن لیست جدید دانشجویان وجود دارد، مرتب‌سازی درجی گزینه مناسبی محسوب می‌شود و از دیگر روش‌های مرتب‌سازی که در این مقاله توضیح داده شده، بهتر عمل می‌کند.

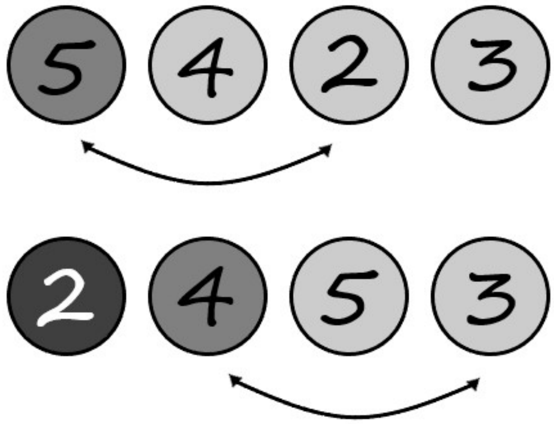
خوب است بدانید می‌توان پیچیدگی زمانی مرتب سازی درجی در بدترین حالت را با تغییراتی در الگوریتم آن بهبود بخشید. به عنوان مثال مرتب‌سازی شِل[۷] یک روش مرتب‌سازی دیگر است که از مرتب‌سازی درجی استفاده می‌کند و در حالت کلی دارای پیچیدگی زمانی بهتری است. [۲] [۳] جزئیات الگوریتم و پیچیدگی زمانی مرتب‌سازی شل را در این مقاله بررسی نمی‌کنیم.

۲ مرتب‌سازی انتخابی[۸]

مرتب‌سازی انتخابی الگوریتم ساده‌ای دارد و احتمالا یکی از اولین الگوریتم‌های مرتب‌سازی باشد که به ذهنمان می‌رسد. مرتب‌سازی انتخابی اینگونه عمل می‌کند که کوچک‌ترین عنصر آرایه را انتخاب می‌کند، آن را در سمت چپ آرایه قرار می‌دهد و سپس به سراغ عناصر باقی‌مانده می‌رود. [۱] برای درک بهتر می‌توانید به شکل ۲ که در ادامه آمده است نگاه کنید.

---

[7] Shell sort
[8] Selection sort



شکل ۲: مراحلی از مرتب کردن اعداد به کمک مرتب‌سازی انتخابی

الگوریتم مرتب‌سازی انتخابی

از اولین عنصر آرایه شروع می‌کنیم. اگر به عنصر i برسیم، i را به عنوان اندیس[۹] کوچک‌ترین عنصر ذخیره می‌کنیم؛ سپس از عنصر بعد از i شروع می‌کنیم و جلو می‌رویم. اگر عنصری از کوچک‌ترین عنصر کوچک‌تر بود اندیس آن را به عنوان اندیس کوچک‌ترین عنصر نگه می‌داریم. وقتی به آخر آرایه رسیدیم، کوچک‌ترین عنصر ذخیره شده را با i جابه‌جا[۱۰] می‌کنیم و به سراغ عنصر بعدی می‌رویم. مراحل گفته شده را برای i جدید تکرار می‌کنیم تا به انتهای آرایه برسیم. شبه‌کد مرتب‌سازی انتخابی در الگوریتم ۲ که در ادامه آمده است قابل مشاهده است. همچنین شبه‌کد جابه‌جایی را می‌توانید در الگوریتم ۳ مشاهده کنید.

[9] index
[10] swap

الگوریتم ۲ مرتب‌سازی انتخابی

```
procedure SelectionSort(arr, n)
    for i ← 0 to n − 1 do
        minIndex ← i
        for j ← i + 1 to n do
            if arr[j] < arr[minIndex] then
                minIndex ← j
        Swap(arr, i, minIndex)
```

الگوریتم ۳ جابه‌جایی

```
procedure Swap(arr, i, j)
    temp ← arr[i]
    arr[i] ← arr[j]
    arr[j] ← temp
```

مثال ۲ *اگر الگوریتم ۲ را روی آرایهٔ [۵, ۴, ۲, ۳, ۷] اجرا کنیم، آرایه بدین صورت مرتب می‌شود:*

$$[۵, ۴, ۲, ۳, ۷] \rightarrow [۲, ۴, ۵, ۳, ۷] \rightarrow [۲, ۳, ۵, ۴, ۷] \rightarrow [۲, ۳, ۴, ۵, ۷] \rightarrow [۲, ۳, ۴, ۵, ۷]$$

مثال ۳ *اگر الگوریتم ۲ را روی آرایهٔ [۷, ۶, ۶, ۴, ۹] اجرا کنیم، آرایه بدین صورت مرتب می‌شود:*

$$[۷, ۶, ۶, ۴, ۹] \rightarrow [۴, ۶, ۶, ۷, ۹] \rightarrow [۴, ۶, ۶, ۷, ۹] \rightarrow [۴, ۶, ۶, ۷, ۹] \rightarrow [۴, ۶, ۶, ۷, ۹]$$

نکته ۱ *توجه کنید در مثال ۳ با اینکه آرایه بعد از یک جابه‌جایی مرتب می‌شود، الگوریتم به کار خود ادامه می‌دهد و به‌ازای تمام عناصر i تا $n - ۱$ اجرا می‌شود.*
با مقایسه مثال ۲ و مثال ۳ که در بالا آمده اند، می‌توان وابسته نبودن مرتب‌سازی انتخابی به عناصر ورودی را بهتر درک کرد.

۳ مرتب‌سازی ادغامی[۱۱]

مرتب‌سازی ادغامی یکی دیگر از روش‌های مرتب سازی است که از روش تقسیم و حل[۱۲] استفاده می‌کند. این روش در سال ۱۹۴۵ میلادی ابداع شد. [۴] استفاده از این روش نسبت به روش‌های قبلی سرعت مرتب شدن اعداد را به طور قابل توجهی بالا می‌برد.

نکته ۲ *مرتب‌سازی ادغامی نسبت به مرتب‌سازی‌های دیگری که در این مقاله بحث شده به حافظه بیشتری نیاز دارد. البته قابل توجه است که اقداماتی برای تغییر این الگوریتم با این هدف که پیچیدگی حافظه آن کم شود انجام شده. چندین روش برای مرتب‌سازی ادغامی با حافظه ثابت[۱۳] یا به عبارتی مرتب سازی ادغامیِ درجا[۱۴] پیشنهاد شده است.* [۵]

الگوریتم ۴ مرتب‌سازی ادغامی

procedure MERGESORT($arr, left, right$)
    if $left < right$ **then**
        $middle \leftarrow \frac{left+right}{2}$
        MERGESORT($arr, left, middle$)
        MERGESORT($arr, middle + 1, right$)
        MERGE($arr, left, middle, right$)

[11] Merge sort
[12] divide and conquer
[13] constant
[14] in-place merge sort

الگوریتم ۵ ادغام

```
procedure Merge(arr, left, middle, right)
    n1 ← middle − left + 1
    n2 ← right − middle
    leftArr ← new Array[n1]
    rightArr ← new Array[n2]
    for i ← 0 to n1 do
        leftArr[i] ← arr[left + i]
    for j ← 0 to n2 do
        rightArr[j] ← arr[middle + 1 + j]
    i ← 0
    j ← 0
    k ← left
    while i < n1 and j < n2 do
        if leftArr[i] ≤ rightArr[j] then
            arr[k] ← leftArr[i]
            i ← i + 1
        else
            arr[k] ← rightArr[j]
            j ← j + 1
        k ← k + 1
    while i < n1 do
        arr[k] ← leftArr[i]
        i ← i + 1
        k ← k + 1
    while j < n2 do
        arr[k] ← rightArr[j]
        j ← j + 1
        k ← k + 1
```

مراجع

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th ed. , 2022.

[2] R. B. Frank, R. M.; Lazarus, “A high-speed sorting procedure,” *Communications of the ACM*, p.20–22, 1960.

[3] R. Sedgewick, “A new upper bound for shellsort,” *Journal of Algorithms*, p.159–173, 1986.

[4] D. Knuth. *The Art of Computer Programming*. Addison-Wesley, 2nd ed. .

[5] M. A. Huang, Bing-Chao; Langston, “Practical in-place merging,” *Communications of the ACM*, p.348–352, 1988.