جزوه جلسه سیزدهم داده ساختارها و الگوریتم

۱۶ ابان ۱۴۰۰

فهرست مطالب

۱ الگوریتم های مرتب سازی خطی

در این بخش از درس، به مطالعه الگوریتم هایی میپردازیم که زمان اجرای آنها کمتر از $O(nlogn)$ میباشد و همچنین در مدل مقایسه (مدل محاسباتی که در جلسه قبل ذکر شد) نیستند. توجه داریم که $logn$ خود مقداری بسیار کوچک در مقایسه با n میباشد و پیدا کردن چنین الگوریتم هایی صرفا از منظر تئوری ارزشمند هستند.
فرض اضافه ای که میکنیم، این است که کلید تمام عناصری که با هم مقایسه میشوند مقداری صحیح و نامنفی است (در مدل مقایسه این فرض را نداشتیم). این فرض معقول است؛ زیرا کلید هایی که با آنها سروکار داریم همواره عدد نیستند و میتوان برای آنها، معادل عددی (یک عدد صحیح نامنفی) درنظر گرفت.
حال فرض میکنیم کلید ها، اعداد صحیح بین 0 تا k-1 هستند.(حالتی که کلید ها منفی هستند نیز معادل همین فرض است). به بیان دیگر برای n شئ (معادلا n کلید) داریم: $k = O(n)$. یعنی کران بالا برای شماره کلید ها، برابر تعداد کلیدها منهای یک است.
همچنین فرض میشود هر کلید در یک کلمه (word) جا میشود.

۱.۱ مرتب سازی شمارشی یا **Counting Sort**

در این الگوریتم برای هر کلید بین 0 تا k-1، یک لیست درنظر میگیریم و هر عنصر را به لیست مربوط به خود اضافه میکنیم. درنهایت با یک پیمایش روی لیست اصلی، محتوای هر لیست که عناصر ما هستند را خروجی میدهیم. کد پایتون زیر شهود بهتری از این الگوریتم ارائه میدهد.

```
1.    def counting _sort (A):
2.        u = 1 + max([x.key for x in A])     #O(n)
3.        D = [[] for i in range(u)]           # O(u)
4.        for x in A:                          # O(n)
5.            D[x.key].append()
6.        i = 0
7.        for chain in D:                      # O(u)
8.            for x in chain:
9.                A[i] = x
10.               i += 1
```

خطوط ده و یازده در مجموع n بار و حلقه خط هفت نیز u بار اجرا میشوند؛ پس میتوان گفت مرتبه زمانی اجرای حلقه خط هفت برابر $O(n+u)$ است.
مرتبه زمانی اجرای الگوریتم برابر $O(n+k)$ (در کد بالا $O(n+u)$) میباشد.
مرتب سازی شمارشی برای k های کوچک قابل استفاده است که این نکته از ضعف های این الگوریتم است. اما به دلیل پایه ای برای الگوریتم های دیگر است؛ ارزشمند و مهم برای یادگیری است.

۲.۱ مرتب سازی مبنایی یا Radix Sort

اساس کار این مرتب سازی، مرتب سازی شمارشی است و عیب آن (کار کردن برای k های کوچک) را مرتفع نموده است و این الگوریتم برای زمانی که k یک چندجمله ای از n باشد ($k = n^{O(1)}$) در زمان خطی کار میکند.
در این الگوریتم فرض میکنیم کلیدها در مبنای b هستند.
d را نیز تعداد ارقام k در مبنای b درنظر میگیریم. ($b^d \simeq k \iff d = log_b k$)
در مرتب سازی از رقم کم ارزش کلید ها شروع به مرتب سازی کرده و به رفم پرارزش میرسیم. هر مرحله از مرتب سازی توسط مرتب سازی شمارشی انجام میشود. نکته این که از هر مرتب سازی پایدار (مرتب سازی که ترتیب عناصر با کلید یکسان قبل و بعد از مرتب سازی یکسان باشد) میتوان به جای counting sort استفاده کرد.
پس با تفاسیر فوق، میتوان زمان اجرای الگوریتم مرتب سازی پایه ای را ارائه کرد. به تعداد d بار مرتب سازی شمارشی اجرا میشود که در این الگوریتم زمان اجرای مرتب سازی شمارش برابر $O(n + b)$ است (هرکلید بین 0 تا b-1 قرار دارد).
پس مرتبه زمانی این الگوریتم برابر است با $O(d(n + b))$.
زمان اجرا در حالت کلی خطی نیست و برای خطی شدن کافی است d از مرتبه $O(1)$ باشد؛ بدین منظور b را برابر n قرار میدهیم و داریم:

$$d = log_b n^{O(1)} = O(1) \implies O((n + b)d) = O((n + n).O(1)) = O(n)$$

کد پایتون مربوط به Radix Sort نیز در شکل زیر آمده است.

```python
def radix_sort(A):
    "Sort A assuming items have non-negative keys"
    n = len(A)
    u = 1 + max([x.key for x in A])             # O(n) find maximum key
    c = 1 + (u.bit_length() // n.bit_length())
    class Obj: pass
    D = [Obj() for a in A]
    for i in range(n):                          # O(nc) make digit tuples
        D[i].digits = []
        D[i].item = A[i]
        high = A[i].key
        for j in range(c):                      # O(c) make digit tuple
            high, low = divmod(high, n)
            D[i].digits.append(low)
    for i in range(c):                          # O(nc) sort each digit
        for j in range(n):                      # O(n) assign key i to tuples
            D[j].key = D[j].digits[i]
        counting_sort(D)                        # O(n) sort on digit i
    for i in range(n):                          # O(n) output to A
        A[i] = D[i].item
```

۳.۱ نکاتی چند در مورد مرتب سازی های خطی

از سوالات حل نشده در مورد مرتب سازی مبنایی میتوان به مرتب سازی برای k های بزرگتر از $n^{O(1)}$ در زمان خطی اشاره کرد.
همچنین در مدل word RAM سریعترین الگوریتم مرتب سازی در زمان $O(n\sqrt{loglogn})$ و با احتمال بالا (مرتب سازی Integer Sort که یک مرتب سازی تصادفی است) میباشد.