پاسخ تمرین شماره 4

Sort & hash

دانشکده مهندسی برق و کامپیوتر

مهلت تحویل:

ساختمان های داده و الگوریتم - پاییز 1400

استاد: دکتر هشام فیلی

1400/9/20, ساعت: 23:59

طراح تمرین : **پارسا کوت‌زری**

سوال اول

$$h(14,\ 0)\ =\ (14\ +\ 0\ +\ 0)\ mod\ 12\ =\ 2$$

$$h(3,\ 0)\ =\ (3\ +\ 0\ +\ 0)\ mod\ 12\ =\ 3$$

$$h(23,\ 0)\ =\ (23\ +\ 0\ +\ 0)\ mod\ 12\ =\ 11$$

$$h(10,\ 0)\ =\ (10\ +\ 0\ +\ 0)\ mod\ 12\ =\ 10$$

$$h(41,\ 0)\ =\ (41\ +\ 0\ +\ 0)\ mod\ 12\ =\ 5$$

جدول نهایی:

address	value
0	
1	
2	14
3	3
4	

5	41
6	
7	
8	
9	
10	10
11	23

سوال دوم

الف) ابتدا عناصر را مرتب میکنیم. هزینه زمانی این کار $O(nlogn)$ است. سپس به ازای هر عنصر مقدار $x - A[i]$ را با استفاده از درخت جستجوی دودویی در لیست مرتب شده جستجو میکنیم که هزینه این کار نیز $O(nlogn)$است. بنابراین هزینه نهایی برابر $O(nlogn)$ است.

ب) از یک جدول درهم‌سازی استفاده کرده و به این صورت عمل میکنیم. از ابتدای آرایه شروع کرده و هر عنصری که میبینیم در hash table ذخیره میکنیم و قبل از رفتن به عنصر بعدی وجود مقدار $x - A[i]$ را در جدول چک میکنیم. اگر چنین مقداری وجود داشته باشد به جواب رسیده‌ایم و در غیر اینصورت همین مراحل را برای عنصر بعدی در آرایه تکرار میکنیم. میدانیم اضافه کردن و چک کردن وجود یک مقدار در جدول هش هر دو با هزینه $O(1)$ انجام می شود بنابراین هزینه کلی برابر $O(n)$ است.

سوال سوم

ابتدا یک جدول درهم‌سازی خالی در نظر میگیریم. سپس عناصر آرایه را پیمایش میکنیم و در مرحله iام ابتدا عنصر فعلی را با متغیر partial_sum که در ابتدای کار صفر است جمع میکنیم و در partial sum ذخیره میکنیم. به این صورت اگر در ایندکس iام باشیم این متغیر مجموع عناصر 0 تا i را در خود دارد. سپس مقدار فعلی partial_sum را به عنوان key و i را به عنوان

value در hash table اضافه میکنیم سپس چک میکنیم آیا کلید partial_sum - k در جدول وجود دارد یا خیر. اگر کلید partial_sum - k وجود داشته باشد و value آن j باشد نتیجه میگیریم عناصری که ایندکس آنها بین i و j است زیر آرایه مورد نظر ما هستند چرا که که مجموع آنها برابر k است. با توجه به اینکه هزینه اضافه کردن یک کلید و چک کردن وجود یک کلید در hash_table برابر $O(1)$ است و این کار برای هر عنصر تکرار می‌شود بنابراین هزینه کلی برابر $O(n)$ است.

سوال چهارم

با توجه به اعداد داده شده بازه اعداد آرایه را صفر تا ۶ در نظر میگیریم و به این صورت آرایه جدید از تعداد تکرار هر عنصر در آرایه میسازیم به این صورت که عدد صفر هیچ بار تکرار نشده، عدد یک ۲ بار تکرار شده، عدد دو ۳ بار تکرار شده و به همین صورت تا اخر سپس در آرایه دیگری شکل تجمیع یافته از آرایه count را محاسبه میکنیم به اینصورت که در ایندکس iام تعداد اعدادی که کوچکتر مساوی i هستند را قرار میدهیم:

$count = [0, 2, 3, 1, 1, 2, 2] \quad cumulative = [0, 2, 5, 6, 7, 9, 11]$

حال تمام عناصر آرایه اصلی را طی میکنیم و با دیدن عنصر x آن را در ایندکس $cumulative[x] - 1$ قرار میدهیم و از مقدار $cumulative[x]$ یکی کم میکنیم آرایه مقصد به این صورت پر خواهد شد.

$result = [-, -, -, -, 2, -, -, -, -, -, -] \quad cumulative = [0, 2, 4, 6, 7, 9, 11]$

$result = [-, 1, -, -, 2, -, -, -, -, -, -] \quad cumulative = [0, 1, 4, 6, 7, 9, 11]$

$result = [-, -, -, -, 2, 3, -, -, -, -, -] \quad cumulative = [0, 1, 4, 5, 7, 9, 11]$

$result = [-, 1, -, -, 2, 3, -, -, -, -, 6] \quad cumulative = [0, 1, 4, 5, 7, 9, 10]$

$result = [-, 1, -, -, 2, 3, -, -, 5, -, 6] \quad cumulative = [0, 1, 4, 5, 7, 8, 10]$

$result = [1, 1, -, -, 2, 3, -, -, 5, -, 6] \quad cumulative = [0, 0, 4, 5, 7, 8, 10]$

$result = [1, 1, -, -, 2, 3, -, 5, 5, -, 6] \quad cumulative = [0, 0, 4, 5, 7, 7, 10]$

$result = [1, 1, -, -, 2, 3, 4, 5, 5, -, 6] \quad cumulative = [0, 0, 4, 5, 6, 7, 10]$

$result = [1, 1, -, 2, 2, 3, 4, 5, 5, -, 6] \quad cumulative = [0, 0, 3, 5, 6, 7, 10]$

$result = [1, 1, -, 2, 2, 3, 4, 5, 5, 6, 6] \quad cumulative = [0, 0, 3, 5, 6, 7, 9]$

$result = [1, 1, 2, 2, 2, 3, 4, 5, 5, 6, 6] \quad cumulative = [0, 0, 2, 5, 6, 7, 9]$

سوال پنجم

آرایه شامل [$A[1]$, $A[2]$, … , $A[n]$] را به صورت آرایه‌ای از زوج مرتب‌ها به شکل [$(A[0], 0)$, $(A[1], 1)$, … , $(A[n], n)$] درمیاوریم. سپس در تابع مقایسه که قرار است در الگوریتم مرتب‌سازی استفاده شود را به اینصورت تعریف میکنیم که اگر عنصر اول دو درایه برابر بود عنصر دوم آنها را با هم مقایسه کند. به اینصورت حالت تساوی که برای آرایه اولیه ما اتفاق می‌افتاد برای آرایه جدید اتفاق نمی‌افتد و در نهایت درایه‌هایی که عنصر اول آنها برابر باشد بر اساس عنصر دوم که همان ایندکس اولیه هر درایه است مرتب میشوند. بنابراین این روش مرتب‌سازی پایدار خواهد بود.

سوال ششم

همایش‌ها را براساس زمان پایان آنها مرتب میکنیم و به ترتیب صعودی در یک آرایه میریزیم. سپس از ابتدای آرایه شروع به پیمایش کرده و هر همایش جدیدی که میبینیم اگر زمان شروع آن از آخرین همایشی که انتخاب کرده‌ایم بزرگ‌تر بود آن را انتخاب میکنیم و در غیر اینصورت به سراغ همایش بعد میرویم(واضح است که اولین همایش در آرایه قطعا انتخاب می‌شود چرا که هنوز هیچ همایشی انتخاب نشده است که با آن تداخل داشته باشد). به این صورت میتوان بیشترین تعدا همایش که قابل برگزاری در این سال است را انتخاب کرد. هزینه‌ی این الگوریتم شامل مرتب‌سازی اولیه با هزینه $O(nlogn)$ و یکبار پیمایش آرایه با هزینه $O(n)$ است. پس در نهایت هزینه کلی برابر $O(nlogn)$ است.

سوال هفتم

برای حل این سوال از ایده الگوریتم merge sort استفاده می کنیم. (این ایده معروف به تقسیم و حل است که در درس طراحی الگوریتم بیشتر با آن آشنا خواهید شد)

ورودی زیرمساله ما یک بازه از آرایه اصلی است و خروجی آن هم آرایه مرتب شده آن بازه و هم تعداد زوج مرتب‌های مطلوب سوال است.

برای حل هر زیرمساله آن را به دو از وسط به دو نیمه مساوی تقسیم می کنیم. مساله را به صورت بازگشتی برای هر کدام از این دو قسمت حل می‌کنیم و حال باید جواب زیرمساله اصلی را از خروجی این دو قسمت محاسبه کنیم. گفتیم که دو خروجی داریم، برای بدست آوردن آرایه مرتب شده دقیقا از الگوریتم merge sort استفاده می کنیم. و اما برای پیدا کردن تعداد زوج

مرتب‌های مطلوب مساله، این زوج ها یا هر دو در یک قسمت هستند که این حالت در زیر مسئله بخش مربوط حل شده است یا اینکه i در نیمه اول و j در نیمه دوم است. نکته سوال پیدا کردن تعداد این زوج ها است.

یک اشاره‌گر روی آرایه مرتب نیمه اول در نظر می‌گیریم و ابتدا تا انتها حرکت می کنیم. به ازای هر عضو می خواهیم همه عضو هایی از نیمه دوم که با این عضو یک زوج مطلوب می سازند را پیدا کنیم. واضح است که این عضو ها در یک بازه متوالی هستند (چون مجموعه اعدادی که در یک آرایه مرتب شده در یک بازه خاص هستند در یک بازه از آرایه هستند، این لم به راحتی با برهان خلف قابل اثبات است) پس دو اشاره‌گر در نیمه دوم برای ابتدا و انتهای این بازه در نظر می‌گیریم و آن را هر مرحله آپدیت می‌کنیم. هر بار که اشاره‌گر نیمه اول جلو می رود به عددی بزرگتر مساوی خواهیم رسید. پس هر کدام از این دو اشاره‌گر در نیمه دوم یا ثابت می‌مانند یا جلو می روند. پس به راحتی تا هنگامی که لازم است آن دو را جلو می بریم. (برای درک بهتر می توانید کد را مشاهده کنید)

حال باید الگوریتم خود را تحلیل زمانی کنیم. چون هر اشاره‌گر در کل یه بار ابتدا تا انتهای آرایه را طی می‌کند پس به صورت سرشکن $O(n)$ هزینه می‌دهیم. پس رابطه بازگشتی به این صورت است:

$$T(n) = 2T(n/2) + O(n) => T(n) = O(nlogn)$$

شبه کد:

بخش شکستن آرایه و سپس مرج کردن آن مانند مرج سورت است و تابع زیر وظیفه شمردن زوج‌های i و j که i در زیر آرایه سمت چپ (L) و j در زیر آرایه سمت راست(R) هست را دارد.

```
count_pairs(L, R, r, l):
    left_pointer = 0;
    first_right_pointer = 0;
    second_right_pointer = 0;
    number_of_pairs = 0
    for left_pointer=0 to length(L):
        while R[first_right_pointer] - L[left_pointer] < r:
            first_right_pointer++;
        while R[second_right_pointer] - L[left_pointer < l:
```

```
                second_right_pointer++;

        number_of_pairs += (second_right_pointer - first_right_pointer)

return number_of_pairs
```