Week 06 Answers

پاسخنامه تکالیف هفته ۰۶

پاسخ سوال اول :

در ابتدا هر عنصر در یک مجموعه جداگانه قرار دارد :
{A} , {B} , {C} , {D} , {E}

مرحله اول ، Union(A, B) :
عناصر A و B در یک مجموعه قرار می گیرند.
{A, B} , {C} , {D} , {E}

مرحله دوم ، Union(A, B) :
عناصر C و D در یک مجموعه قرار می‌گیرند.
{A, B} , {C, D} , {E}

مرحله سوم ، Union(B, C) :
از آنجا که B در مجموعه {A, B} و C در مجموعه {C, D} قرار دارد، این دو مجموعه باهم ادغام می‌شوند.
{A, B, C, D} , {E}

پاسخ نهایی :
تعداد مجموعه‌های مجزا: ۲
مجموعه‌ها : {A, B, C, D} , {E}
A و D در یک مجموعه هستند ؟ بله

پاسخ سوال دوم :

۱) ساختار Disjoint Set چیست؟

ساختاری برای مدیریت گروه‌های جدا از هم با دو عملیات اصلی:

- Find(x) : پیدا کردن نماینده گروهی که x در آن است
- Union(x, y) : ادغام گروه x با گروه y

۲) نماینده گروه چیست ؟

هر گروه به صورت درخت ذخیره می‌شود. ریشه درخت = نماینده گروه

A (نماینده)
|
B
|
C

Find(C) : از A بالا می‌رود تا به C برسد.

۳) عملیات MakeSet :

هر کاربری که برای اولین بار ظاهر می شود :

parent[x] = x (خودش نماینده خودش است)

۴) الگوریتم Find پایه :

```
while parent[x] != x:
  x = parent[x]
return x
```

۵) الگوریتم Union پایه :

```
ra = Find(a)
rb = Find(b)
if ra != rb:
   parent[rb] = ra
```

۶) حل مثال :

وضعیت اولیه: {A} {B} {C} {D}
CONNECT A B → {AB} {C} {D}
CONNECT C D → {AB} {CD}
CHECK A C → Find(A)=A, Find(C)=C → مختلف → NO
CONNECT B C → {ABCD}
CHECK A D → Find(A)=A, Find(D)=A → یکسان → YES

پاسخ سوال سوم :

از آنجا که آرایه مرتب است، می‌توان از Binary Search استفاده کرد.

۱) چرا Binary Search لازم است؟
آرایه مرتب است. پس می‌توانیم به جای چک کردن تک‌تک عناصر($O(n)$)، با Binary Search به جواب برسیم. ($O(\log n)$)

در این سؤال هم تاکید داریم جستجوی خطی مجاز نیست.

۲) مشکل اصلی: پیدا کردن "اولین/آخرین" وقوع
Binary Search معمولی فقط می‌گوید x هست یا نیست، اما برای اولین/آخرین باید کمی تغییر ایجاد کنیم.
در اینجا از دو تابع استاندارد استفاده می‌کنیم:

lower_bound(x)
اولین اندیس i که:
arr[i] >= x

upper_bound(x)
اولین اندیس j که:
arr[j] > x

۳) چطور از این‌ها ، اولین/آخرین وقوع در می‌آید؟
اگر x در آرایه وجود داشته باشد:
first = lower_bound(x)
last = upper_bound(x) - 1
چرا؟
چون upper_bound(x) می‌رود بعد از آخرین x .پس یکی قبلش می‌شود آخرین x.

4) تشخیص اینکه x اصلاً وجود دارد یا نه.
ممکن است lower_bound(x) یک اندیس بدهد، ولی مقدار آن x نباشد (مثلاً نزدیک‌ترین بزرگ‌تر باشد).
پس چک می‌کنیم:
اگر first == n یا arr[first] != x و وجود ندارد ← -1 -1

5) پیچیدگی زمانی و فضایی

lower_bound:
$O(logn)$
upper_bound:
$O(logn)$

کل :
$O(logn)$
فضای اضافی :
$O(1)$
برای یافتن اولین وقوع x ، از lower_bound استفاده می‌شود که اولین اندیس عنصر بزرگ‌تر یا مساوی x را پیدا می‌کند.
برای یافتن آخرین وقوع x، از upper_bound استفاده می‌شود و یک واحد از اندیس به‌دست‌آمده کم می‌کنیم.
اگر lower_bound خارج از محدوده آرایه باشد یا مقدار آن برابر x نباشد، عدد x در آرایه وجود ندارد.

پاسخ سوال چهارم :

برای هر پرس‌وجو از دو جستجوی دودویی استفاده می‌شود:

یک آرایه‌ی مرتب داریم و چند Query .
۱) هر Query می‌پرسد:
چند عنصر آرایه ،داخل بازه‌ی شامل دو سر یعنی [L, R] قرار دارند؟
۲) چرا جستجوی خطی ممنوع/بد است؟
اگر برای هر Query آرایه را کامل بگردیم، می‌شود:
هر Query :
$O(n)$
کل :
$O(nq)$
اگر n و q بزرگ باشند، خیلی کند می‌شود.

۳) ایده‌ی اصلی : مرزهای بازه را با Binary Search پیدا کن.
چون آرایه مرتب است، می‌توانیم جایگاه مرزها را سریع پیدا کنیم :
lower_bound(L)
اولین اندیس عنصر بزرگ‌تر یا مساوی.L
upper_bound(R)
اولین اندیس عنصر بزرگ‌تر از R .
پس تمام عناصر داخل بازه دقیقاً بین این دو اندیس قرار می‌گیرند.

۴) فرمول شمارش
تعداد عناصر در بازه [L, R] برابر است با:
count=upper_bound(R)-lower_bound(L)
چرا؟
lower_bound(L) می‌گوید از کجا شروع می‌شوند (اولین عنصر قابل قبول)
upper_bound(R) می‌گوید تا کجا ادامه دارند (اولین عنصر خارج از بازه)
اختلافشان تعداد عناصر داخل بازه است.

۵) حالت‌های مرزی مهم
اگر R از همه عناصر کوچک‌تر باشد upper_bound(R) ممکن است ۰ شو ← شمارش ۰
اگر L از همه عناصر بزرگ‌تر باشد lower_bound(L) = n ← شمارش ۰
تکرارها کاملاً درست حساب می‌شوند چون lower/upper برای آرایه‌ی غیرکاهشی طراحی شده‌اند.

۶) پیچیدگی زمانی
هر Query دو بار Binary Search دارد
$O(logn)$
برای q Query
$O(qlogn)$

فضای اضافی :
$O(1)$ غیر از نگهداری ورودی
lower_bound(L) : اولین اندیس عنصر بزرگ‌تر یا مساوی L
upper_bound(R) : اولین اندیس عنصر بزرگ‌تر از R
تعداد عناصر داخل بازه برابر است با:
upper_bound(R) - lower_bound(L)
این روش باعث می‌شود هر پرس‌وجو در زمان $O(log\ n)$ پاسخ داده شود.

پاسخ سوال پنجم :

برای حل این مسئله از الگوریتم Median of Medians استفاده می‌شود.

می‌خواهیم k اُمین عنصر کوچک‌تر را پیدا کنیم، بدون اینکه کل آرایه را مرتب کنیم.
مرتب‌سازی کامل $O(nlogn)$ است، ولی selection می‌تواند خطی باشد.
چرا Quickselect معمولی کافی نیست؟
Quickselectبا pivot تصادفی/ساده،به طور میانگین خوب است، اما در بدترین حالت ممکن است $O(n^2)$شود.
Median of Medians چه می‌کند؟
یک pivot به‌اندازه کافی خوب انتخاب می‌کند تا هر بار بخش قابل توجه از عناصر حذف شوند.
مراحل:
تقسیم آرایه به گروه‌های ۵تایی
گرفتن median هر گروه با sort گروه کوچک
پیدا کردن «میانه‌یmedian ها» به صورت بازگشتی ← این می‌شودpivot
آرایه را به $pivot <$ و $pivot >$ و خود $pivot$ تقسیم می‌کنیم
بسته به اینکه k کجاست، فقط روی همان بخش recurse می‌کنیم
چرا بدترین حالت $O(n)$ می‌شود؟

به‌صورت شهودی pivot :طوری انتخاب می‌شود که همیشه تعداد خوبی از عناصر حتماً از pivot کوچک‌تر و تعداد خوبی حتماً بزرگ‌تر باشند؛ پس اندازه‌ی مسئله سریع کوچک می‌شود و مجموع هزینه‌ها خطی می‌ماند.

پیچیدگی:

ساخت median ها و partition در هر سطح :

$O(n)$

عمق بازگشت به خاطر pivot خوب کنترل می‌شود

نتیجه: بدترین حالت

$O(n)$