# الگوریتم‌های پیشرفته

## راه‌حل تمرین سری اول

---

1. فرض کنید یک آرایه نامرتب از اعداد صحیح در بازه‌ی صفر تا n به شما داده شده است، از میان اعداد داده شده یک عدد مفقود شده است، الگوریتمی با پیچیدگی O(n) و با رویکرد تقسیم و غلبه برای پیدا کردن این عدد ارائه دهید، در مورد پیچیدگی الگوریتم خود نیز بحث کنید.

راه‌حل:

می‌توانیم از SELECT برای یافتن عنصر median استفاده کنیم و بررسی کنیم که آیا در آرایه است یا خیر. اگر اینطور نیست، آن عدد گم شده است، در غیر این صورت، آرایه را در اطراف عنصر میانه x به دو گروه کوچکتر و مساوی x و بزرگتر از x تقسیم می‌کنیم، اگر اندازه اولی کمتر از x+1 باشد، در این زیر آرایه مجدداً همین عمل را انجام می‌دهیم. در غیر این صورت ما در زیر آرایه دیگر بازگشت می کنیم.

The procedure MISSINGINTEGER$(A, n, [i, j])$ takes as input an array $A$ and a range $[i, j]$ in which the missing number lies.

```
MISSINGINTEGER(A, [i, j])
1  Determine median element x in range i...j
2  Check to see if x is in A
3  PARTITION A into B, elements < x, and C, elements ≥ x
4  If SIZE(B) < x + 1
5     MISSINGINTEGER(B, [i, x])
6  Else MISSINGINTEGER(C, [x + 1, j])
```

The running time is $O(n)$ because the recurrence for this algorithm is $T(n) = T(n/2) + n$, which is $O(n)$ by the Master Method.

2. در یک آرایه دلخواه ( نامرتب ) الگوریتمی برای پیدا کردن یک مینیمم محلی، با پیچیدگی O(lg n) ارائه دهید.

راه‌حل:

یک راه حل ساده این است که یک اسکن خطی از آرایه انجام دهیم و به محض اینکه یک مینیمم محلی پیدا کردیم، آن را برگردانیم. بدترین حالت پیچیدگی زمانی این روش O(n) خواهد بود. یک راه حل کارآمد مبتنی بر جستجوی باینری است. عنصر میانی را با همسایگانش مقایسه می کنیم. اگر عنصر میانی از هیچ یک از همسایگان خود بزرگتر نباشد، آن را برمی گردانیم. اگر عنصر میانی بزرگتر از همسایه چپ خود باشد، در این صورت همیشه یک حداقل محلی در نیمه چپ وجود دارد. اگر عنصر میانی بزرگتر از همسایه سمت راست خود باشد، در این صورت همیشه یک حداقل محلی در نیمه راست وجود دارد .

3. یک آرایه از اعداد طبیعی به شما داده شده است، الگوریتمی با رویکرد تقسیم و غلبه و پیچیدگی O(lg n) طراحی کنید که تعداد اعداد کوچکتر از هر عدد که ایندکس آنها در آرایه بزرگتر از ایندکس عدد فعلی باشد را مشخص کند، برای مثال،

Input : [8, 9, 2, 3, 4, 1]

Output : [4, 4, 1, 1, 1, 0]

راه‌حل:

برای حل این سوال از ایده merge sort استفاده می‌کنیم به این معنی که با ایجاد تغییرات اندکی در هنگام مرتب‌سازی، خروجی را ایجاد می‌کنیم، یک شمارنده به نام counter میسازیم همچنین در هنگام merge کردن دو زیر مساله یک نشانگر بر روی زیر مساله اول به

نام low و یک نشانگر بر روی ابتدای زیر مساله دوم به نام high قرار می‌دهیم، اگر موقعیتی که low به آن اشاره دارد از نظر مقداری کمتر از موقعیتی که high به آن اشاره دارد باشد conter را یک واحد افزایش می‌دهیم و اشاره‌گر high را جلو میبریم همین عمل را تکرار میکنیم اگر مقدار ایندکس low از high بزرگتر شد آنگاه در آرایه جواب مقدار ایندکس مربوطه را با counter جمع می‌کنیم و اشاره‌گر low را جلو می‌بریم به اینصورت در نهایت در آرایه جواب که ابتدا همه عناصر آن صفر است، جواب نهایی را قرار می‌گیرد.

برای مطالعه بیشتر و دیدن کد به لینک مراجعه کنید.

همچنین مسئله بسیار شبیه به محاسبه تعداد inversionها می باشد.

4. یک لیست پیوندی را در نظر بگیرید که عملیات‌های زیر را در آن تعریف شده است ؛

Insert (x) : Adds the element x to the end of the list

oddDelete() : Removes every element at a location which is an odd number in the list. i.e. removes the first, third, fifth, etc., elements of the list.

فرض کنید این عملیات insert دارای هزینه 1 و عملیات oddDelete دارای هزینه‌ای معادل با هزینه برابر با تعداد عناصر موجود در لیست است

a. فرض کنید n عملیات را انجام می‌دهیم. بدترین زمان فراخوانی oddDelete چیست؟
b. هزینه‌ی سرشکن هر یک از عملیات insert و oddDelete را با استفاده از روش accounting نشان دهید.

c. حال با استفاده از روش تابع پتانسیل هزینه‌ی سرشکن هر یک از عملیات insert و oddDelete را به‌دست آورید.

a) بدترین حالت O(n) است که زمانی اتفاق می افتد که Insert() را n-1 بار و سپس oddDelete را فراخوانی کنیم.

b) درج 3 تومان شارژ می شود. oddDelete نیز 0 تومان شارژ می شود. وقتی insert فراخوانی می‌شود، بلافاصله 1 دلار برای پرداخت هزینه فراخوانی خرج می‌کنیم، سپس دو دلار باقی‌مانده را با آیتم اضافه شده به لیست ذخیره می‌کنیم. وقتی oddDelete را فراخوانی می کنیم، هر عنصر موجود در لیست دو تومان با خود ذخیره می کند. ما از هر عنصر یک تومان می گیریم تا هزینه فراخوانی oddDelete را بپردازیم. علاوه بر این، به ازای هر مورد حذف شده از لیست، تومان اضافی ذخیره شده در آن عنصر را می گیریم و آن را در مورد بعدی لیست قرار می دهیم. بنابراین، در پایان فراخوانی به oddDelete، همه عناصر موجود در لیست هنوز دو تومان روی آنها ذخیره شده است. بنابراین هزینه مستهلک شده به ازای هر عملیات O(1) است.

c) فرض کنید $L_i$ لیست بعد از عملیات i_ام باشد و $num(S_i)$ تعداد عناصر موجود در $L_i$ باشد تابع پتانسیل را به‌صورت $\varphi_i = 2 * num(L_i)$ در نظر بگیرید) تعداد موارد موجود در لیست ابتدا 0 است و همیشه غیرمنفی است، بنابراین φ معتبر است. ابتدا هزینه مستهلک شده درج در زمان i را محاسبه می کنیم. می‌دانیم که

$c_i=1$ همچنین بنابراین $a_i=3$ است، سپس هزینه مستهلک شده oddDelete را محاسبه می کنیم، می‌دانیم که $c_i=num(S_{i-1})$ همچنین $\varphi_i - \varphi_{i-1} = -num(S_{i-1})$ بنابراین $a_i=0$ می‌شود که این نشان می دهد که هزینه مستهلک شده هر دو عملیات O(1) است. یادآوری:

$$a_i = c_i + \varphi_i - \varphi_{i-1}$$

5. فرض کنید در حال ایجاد یک ساختار داده آرایه‌ای هستید که دارای اندازه ثابت n است. می‌خواهیم پس از هر تعداد عملیات درج، از این آرایه یک نسخه پشتیبان تهیه کنیم. متأسفانه، عملیات پشتیبان‌گیری بسیار پرهزینه است، انجام این عملیات دارای پیچیدگی O(n) است. درج بدون پشتیبان فقط 1 واحد زمان می برد.
   a. چگونه می‌توان یک نسخه پشتیبان تهیه کرد و همچنان تضمین داشته باشیم که هزینه سرشکن شده درج O(1) است؟
   b. ثابت کنید که می‌توانی عملیات پشتیبان‌گیری را در زمان سرشکن شده O(1) انجام داد. از روش پتانسیل برای اثبات خود استفاده کنید.

راه‌حل:

A. می توانید پس از هر n درج از آرایه پشتیبان تهیه کنید

B.

$$\varphi_i = i \bmod n$$
When $i \bmod n = 0 \rightarrow a_i = n + 0 - (n-1) = 1$
When $i \bmod n \neq 0 \rightarrow a_i = 1 + (i \bmod n) - \big((i-1)\big) \bmod n = 2$

6. فرض کنید یک "شمارنده سه‌گانه" به این صورت تعریف شده که هر یک از ارقام آن 1 ، 0 یا 1- میتواند باشد و ارقام آن مشابه شمارنده‌ی دودویی مبنای 2 هستند. به‌طور مثال عدد 3 را در این شمارنده میتوان به دو صورت 11 و 10(-1) نشان داد. عمل افزایش در این شمارنده مشابه شمارنده‌ی دودویی بوده و به کم ارزش‌ترین رقم 1 را اضافه میکنیم. اگر این رقم دو شد، آن را صفر کرده و یکی به رقم بعدی اضافه میشود و این کار ادامه مییابد تا این که رقمی که به آن اضافه شده کمتر از 2 شده باشد. عمل کاهش در این شمارنده نیز به همین صورت است. عدد 1 را از کم ارزش‌ترین رقم کم میکنیم، اگر 2- شد، آن را صفر کرده و از رقم بعدی یکی کم میکنیم، تا جایی که رقم حاصل بزرگتر از 2- باشد، با استفاده از روش پتانسیل، هزینه‌ی سرشکن عمل افزایش و کاهش در این شمارنده را محاسبه کنید.

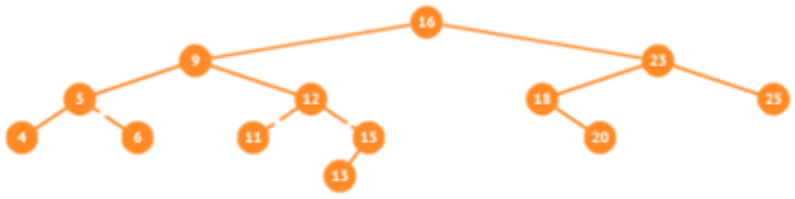
راه‌حل:

تابع پتانسیل را به‌صورت تعداد ارقام غیر صفر شمارنده در نظر می‌گیریم. هزینه تغییر هر رقم در این شمارنده را نیز برابر ۱ درنظر می‌گیریم. برای عمل افزایش براساس صورت سوال حداکثر یک رقم صفر در شمارنده تغییر خواهد کرد. فرض کنید این عمل منجر به تغییر k رقم غیرصفر و m رقم صفر در شمارنده شود، که $m \leq 1$ در نتیجه هزینه‌ی کلی این تغییرات برابر $k + m$ خواهد بود. همچنین تعداد تغییرات ایجاد شده در تابع پتانسیل به‌صورت $\Delta\Phi = m - k$ خواهد بود. درنتیجه:

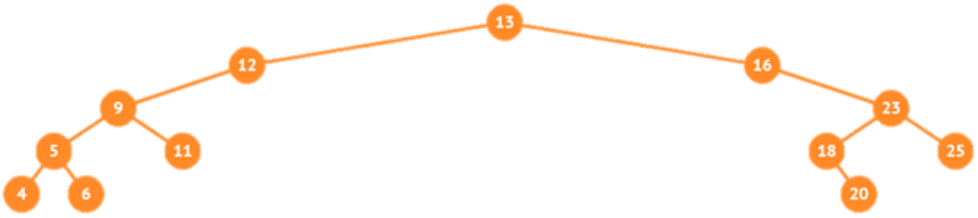
$$cost_{am} = cost_{real} + \Phi(D_i) - \Phi(D_{i-1}) = (k + m) + (m - k) = 2m \leq 2$$

برای عمل کاهش نیز دقیقا به همین صورت هزینه سرشکن قابل محاسبه است و هزینه سرشکن عمل افزایش و کاهش در این شمارنده حداکثر ۲ خواهد بود.

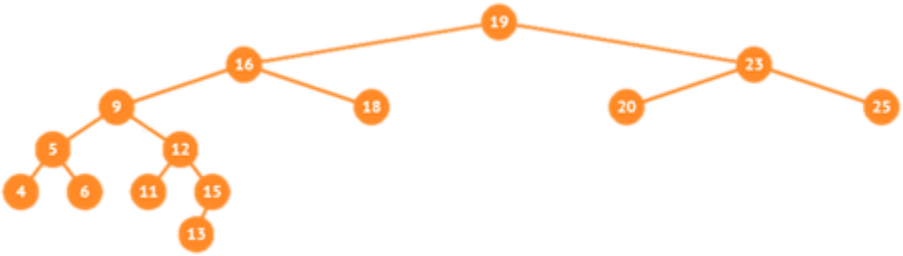
7. درخت Splay زیر را در نظر بگیرید و عملیات‌های خواسته شده را بر روی آن انجام دهید، نحوه انجام هر عملیات را به صورت مرحله به مرحله نمایش دهید.



a. حذف عنصر 15




b. اضافه کردن عنصر 19




c. عملیات split با توجه به عنصر 13