مسئله‌ی ۱. دیکشنری با درخت

یک دیکشنری مرتب شده را که به وسیله درخت AVL پیاده سازی شده است، در نظر بگیرید. نشان دهید که می توان عملیات زیر را، با تغییر اندک در ساختمان داده مذکور، در زمان $\mathcal{O}(\log n)$ پشتیبانی کرد:

countAllinRange(k_1, k_2): تعداد تمامی عنصرهایی مانند k که بین k_1 و k_2 اند را بیاب.

مسئله‌ی ۲. پیوند دو زیر درخت

فرض کنید که دو درخت AVL با نام های T_1 و T_2 داده شده است. به طوری که بزرگترین عنصر در T_1 از کوچکترین عنصر در T_2 مقدار کمتری دارد. عملیات Join(T_1, T_2) را تعریف می کنیم به طوری که حاصل آن یک درخت AVL شامل اجتماع عناصر دو درخت باشد. الگوریتمی ارائه دهید که در زمان $\mathcal{O}(\log n)$، به طوری که n تعداد عناصر درخت حاصل شده است، عملیات فوق را انجام دهد.

مسئله‌ی ۳. متوسط مقادیر درخت

الگوریتمی ارائه دهید که بتوان در زمان ثابت میانگین عناصر یک زیر درخت را به ریشه گره x در یک درخت AVL مانند T محاسبه کرد. لازم است تا الگوریتم شما مشخص کند در هر چرخش چگونه در زمان ثابت می توان گره ها را به روز کرد تا مقدار میانگین را بتوان کماکان محاسبه نمود.

مسئله‌ی ۴. یک پیوند دیگر!

عملیات **join** دو مجموعه غیر ایستای S_1 و S_2 و عنصر x را در نظر می گیرد، به طوری که $\forall x_1 \in S_1$ و $\forall x_2 \in S_2$ داریم که $key[x_1] \leqslant x \leqslant key[x_2]$. حاصل این عملیات، مجموعه $S = S_1 \cup \{x\} \cup S_2$ خواهد بود. فرض کنید که یک درخت قرمزـ سیاه به نام T داده شده است. همچنین مفروض است که تعداد گره های سیاه در هر مسیر از ریشه هر زیر درخت T' به ریشه درخت T در متغیری به نام $bh[T']$ ذخیره شده است. اکنون قصد داریم عملیات **join**(T_1, x, T_2) را پیاده سازی کنیم، به طوری که درخت های قرمزـ سیاه T_1 و T_2 را منحل کرده و حاصل درخت $T = T_1 \cup \{x\} \cup T_2$ خواهد بود. نیز، n را تعداد گره های T در نظر بگیرید.

الف) مفروض است که $bh[T_1] > bh[T_2]$. الگوریتمی با زمان اجرای $\mathcal{O}(\log n)$ ارائه دهید که یک گره سیاه به نام y را در T_1 با بزرگترین کلید در میان گره هایی پیدا می کند که پارامتر bh شان برابر $bh[T_2]$ است.

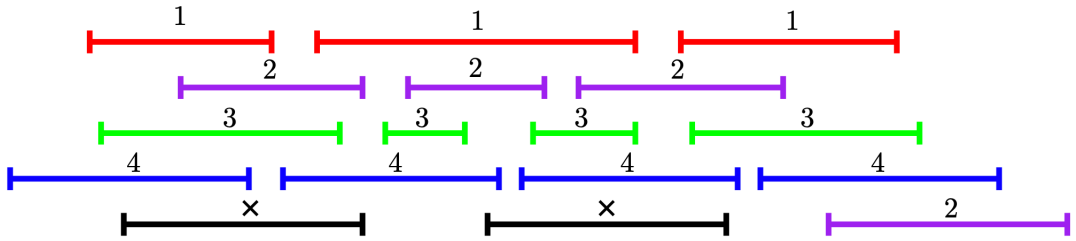
ب) نشان دهید که چگونه $T_y \cup \{x\} \cup T_2$ را که در آن T_y زیر درخت با ریشه y است، می توان در زمان ثابت جایگزین T_y نمود، بدون آنکه خاصیت درخت دودویی از بین برود.

ج) چه رنگی را می بایست به گره x نسبت دهیم تا خاصیت درخت قرمزـ سیاه در قسمت های بالای آن نیز حفظ شود؟ (فرض کنید همه برگ ها رنگ سیاه دارند)

د) نشان دهید که زمان اجرای عملیات **join**(T_1, x, T_2) از مرتبه $\mathcal{O}(\log n)$ است.

مسئله‌ی ۵. رنگ آمیزی بیشترین تعداد بازه

تعداد n بازه به فرم $[s_i, f_i]$ داده شده که هر s_i و f_i نمایانگر یک عدد صحیح مثبت بوده و نیز $s_i < f_i$. همچنین، یک عدد صحیح مثبت به نام k نیز عضوی از ورودی مسئله است. هدف رنگ آمیزی بیشترین تعداد بازه با استفاده از k عدد رنگ است. به نحوی که هر بازه نهایتا یک رنگ دریافت کرده و بازه هایی که همپوشانی دارند، رنگ یکسان نداشته باشند. لازم به ذکر است که دو بازه $[s_i, f_i]$ و $[s_j, f_j]$ همپوشانی دارند، چنانچه $[s_i, f_i] \cap [s_j, f_j] \neq$. الگوریتمی طراحی کنید که در زمان $\mathcal{O}(n \log n)$ چنین رنگ آمیزی‌ای را بیابد.



شکل ۱: رنگ آمیزی بازه‌های فوق با چهار رنگ. بازه‌های ضربدر خورده رنگ‌آمیزی نشده‌اند.

مسئله‌ی ۶. نوعی درخت دودویی

یک درخت دودویی T را در نظر بگیرید. تعداد گره‌های درون T را با $|T|$ نمایش می‌دهیم. برای یک گره مانند $x \in T$، L_x زیردرخت سمت چپ آن و R_x زیر درخت سمت راستش خواهد بود. گوییم زیر درخت با ریشه x تقریبا متوازن است، ABP(x)، اگر و تنها اگر $|L_x| \leqslant ۲|R_x|$ و $|R_x| \leqslant ۲|L_x|$.

الف) بیشترین ارتفاع یک درخت دودویی T با n گره در حالی که ABP$(root)$ برقرار است، چقدر خواهد بود؟

ب) به یک درخت ABP گوییم چنانچه خاصیت مذکور برای تمامی گره های درخت برقرار باشد. نشان دهید که اگر چنین بود، ارتفاع درخت $\mathcal{O}(\log n)$ است. به طور دقیق‌تر، نشان دهید که برای یک درخت ABP مانند T،

$$\text{height}(T) \leqslant \log_{۲} n / \log_{۲} ۳/۲$$

مسئله‌ی ۷. WB-BST

برای هر گره u در یک درخت ریشه دار، size(u) را تعداد گره‌های موجود در زیر درخت به ریشه u در نظر بگیرید. WB-BST[۱] یک درخت دودویی است که ناوردایی زیر در آن برقرار است:

$$size(\text{u.left}) \leqslant \frac{۲}{۳} \cdot size(u) \text{ and } size(\text{u.right}) \leqslant \frac{۲}{۳} \cdot size(u)$$

که در آن u.right فرزند راست u و u.left فرزند چپ آن است. هنگامی که بابت درج و یا حذف گره‌ها ناوردایی بهم بخورد، ایتدا گره‌ای که بیشترین ارتفاع را داشته و ناوردایی در آن صدق نمی‌کند، در نظر گرفته می‌شود. سپس زیر درخت آن گره به طور کامل از نو ساخته می‌گردد، به نحوی که اختلاف ارتفاع زیردرخت راست و چپ هر گره در آن زیر درخت، حداکثر یک باشد.

الف) نشان دهید که ارتفاع درخت هنگامی که n گره را در خود دارد، $\mathcal{O}(\log_{۳/۲} n)$ است.

ب) نشان دهید که بازسازی زیردرخت یک گره مانند u، حداکثر $\mathcal{O}(size(u))$ زمان خواهد برد.

ج) نشان دهید که بعد از عملیات بازسازی، توازن درخت برقرار شده است.

[۱] Weight-balanced Binary Search Tree

د) با روش حسابداری، نشان دهید که هزینه سرشکن هر درج و یا حذف از مرتبه زمانی $\mathcal{O}(\log n)$ است.

ه) مورد بالا را با استفاده از تابع پتانسیل نیز ثابت کنید.

مسئله‌ی ۸. اطلاعات اضافه

فرض کنید که در هر گره x از یک tree ۴-۳-۲ فیلد جدیدی به نام $height[x]$ وجود دارد که ارتفاع زیر درخت به ریشه x را در خود ذخیره کرده است.

الف) نشان دهید چگونه می‌توان عملیات درج و حذف را در این ساختمان داده تغییر داد تا کماکان با مرتبه زمانی $\mathcal{O}(\log n)$ هم عملیات درج و حذف و هم به روز رسانی فیلد اضافه شده به هر گره انجام شود.

ب) با استفاده از قسمت قبل، یک الگوریتم با مرتبه زمانی $\mathcal{O}(۱ + |h_۱ - h_۲|)$ ارائه دهید تا عملیات **join** را بر روی دو tree ۴-۳-۲ با ارتفاع $h_۱$ و $h_۲$ انجام دهد. عملیات **join** دو درخت $T_۱$ و $T_۲$ و عنصر x را گرفته، به نحوی که $\forall y_۱ \in T_۱$ و $\forall y_۲ \in T_۲$، داریم که $key[y_۱] < x < key[y_۲]$. خروجی این عملیات درختی جدید شامل تمامی عناصر $T_۱$ و $T_۲$ و گره x است.

ج) الگوریتمی از مرتبه زمانی $\mathcal{O}(\log n)$ پیشنهاد دهید که عملیات **split** را انجام دهد. این عملیات دقیقا برعکس عملیات **join** عمل می‌کند. الگوریتم شما می‌بایست یک درخت و یک کلید k به عنوان ورودی دریافت نماید. می‌توانید از مسیری که از ریشه درخت تا گره حاوی کلید k در الگوریتم خود کمک بگیرید. فرض کنید مسیر مذکور در خود کلیدهای $\{k_۱, k_۲, \cdots, k_m\}$ را جا داده است. مشخصا، می‌توانید زیر درخت راست و چپ هر کلید k_i و رابطه آنها با k را در نظر بگیرید. همچنین مجازید از عملیات **join** در طراحی الگوریتم خود استفاده کنید.

مسئله‌ی ۹. ساختن Heap

الگوریتم زیر را برای ساختن Heap به نام H در نظر بگیرید. این الگوریتم ابتدا آرایه A را با n عنصر دریافت کرده و سپس با یافتن عنصر minimal به نام $x \in A$ آن را ریشه H قرار می دهد. سپس به طور بازگشتی دو زیر Heap گره x را می‌سازد. که هر یک تقریبا $\frac{n-۱}{۲}$ تا عنصر دارد. یک تحلیل زمانی برای الگوریتم مذکور ارائه دهید. همچنین، راهی برای بهبودالگوریتم داده شده ارائه کنید.

SlowHeap(i, j)

If $i = j$ then return pointer to heap consisting of node containing $A[i]$
Find $i \leq l \leq j$ such that $x = A[l]$ is the minimum element in $A[i \ldots j]$
Exchange $A[l]$ and $A[j]$
$Ptr_{\text{left}} =$ SlowHeap$(i, \lfloor \frac{i+j-1}{2} \rfloor)$
$Ptr_{\text{right}} =$ SlowHeap$(\lfloor \frac{i+j-1}{2} \rfloor + 1, j - 1)$
Return pointer to heap consisting of root r containing x with child pointers Ptr_{left} and Ptr_{right}

End