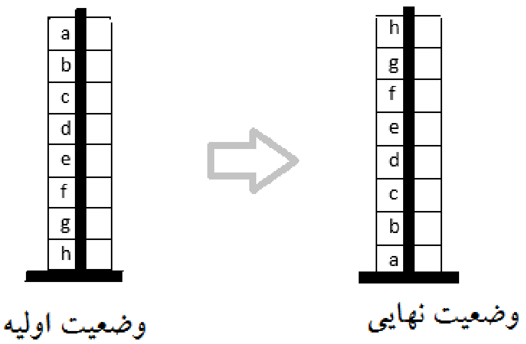
جواب تکلیف سری سوم

ساختمان داده ها و الگوریتمها
دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی. پاییز ۱۴۰۲

۱. **وارون کردن محتویات برج هانوی.** اینجا بر خلاف مسئله برج هانوی، دیسکها اندازه ندارند و هر دیسک فقط یک شماره منحصر بفرد دارد. دیسکهای با شماره های مختلف را می‌توان روی هم قرار داد. می‌خواهیم با کمترین میزان جابجایی محتوای یک برج هانوی وارون شود.



این کار را می‌خواهیم با داشتن محدودیتهای مختلف انجام دهیم. برای هر یکی از موارد زیر یک الگوریتم ارائه دهید و زمان اجرای آن را ذکر کنید. زمان اجرا برابر با جابجایی هایی است که انجام می‌دهید.

(آ) مجاز هستیم از یک برج هانوی دیگر و به تعداد $O(1)$ دیسک روی زمین قرار دهیم.

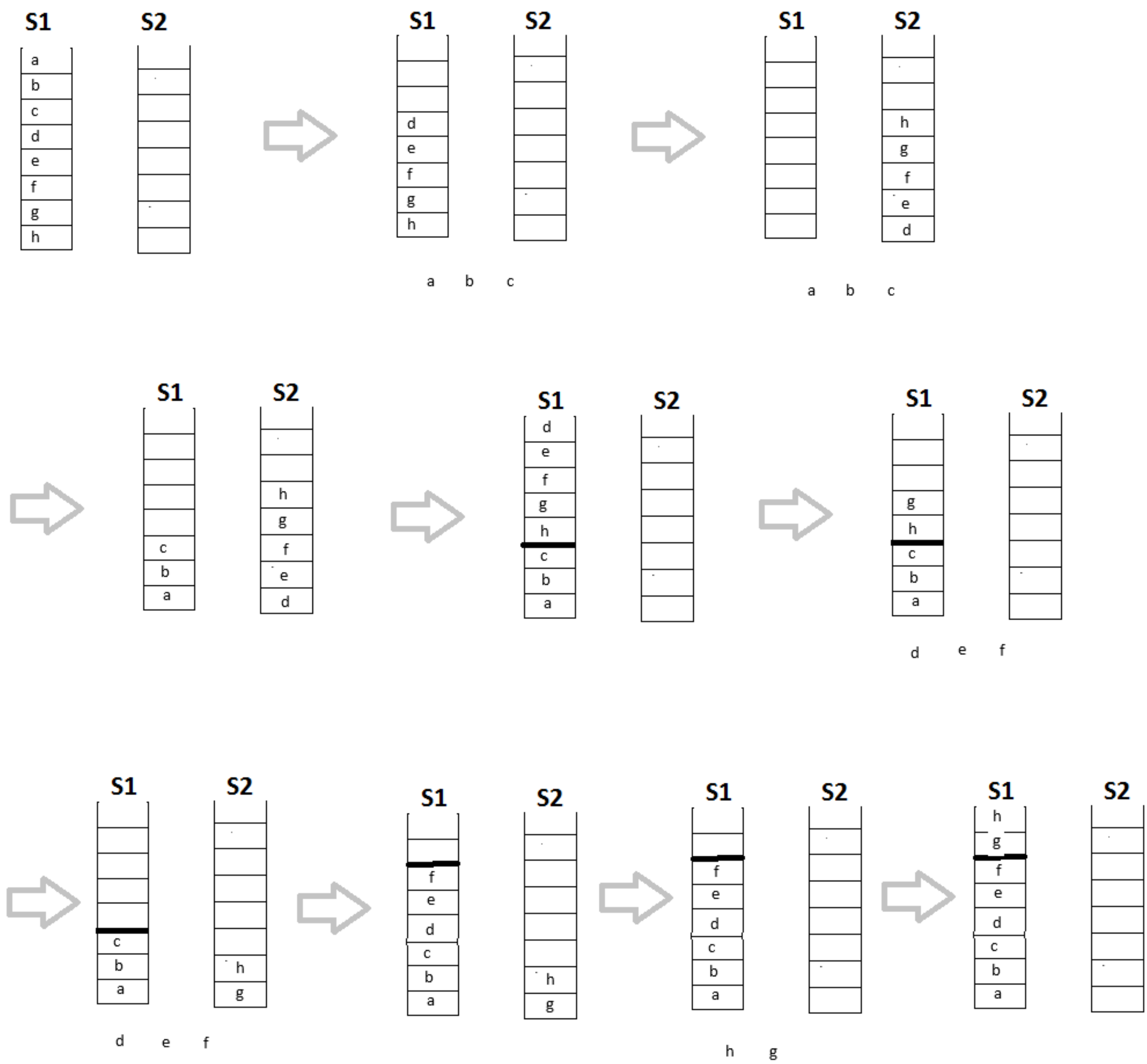
فرض کنید برج هانوی اصلی $S1$ و برج هانوی کمکی $S2$ است. اگر دیسکها را یکی یکی از بالای برج $S1$ حذف کنیم و در برج $S2$ درج کنیم، در نهایت برج $S2$ وارون برج $S1$ خواهد شد. برای رسیدن به هدف، مثل مرتب سازی انتخابی، در هر مرحله یکی از دیسکهای برج$S1$ را به جای درست خود منتقل می‌کنیم. برای نمونه در مثال بالا اول a را به کف برج $S1$ می‌بریم. برای انجام این کار ابتدا دیسک a را روی زمین می‌گذاریم و عناصر زیر آن را به برج کمکی $S2$ منتقل میکنیم. حال a را در کف برج خالی $S1$ قرار می‌دهیم و دیگر کاری به آن نداریم. حال عناصر برج $S2$ را به $S1$ برمی‌گردانیم. دوباره همین کار را تکرار می‌کنیم تا دیسک بعدی (b) بالای a قرار گیرد. به همین ترتیب بقیه دیسکها را در جای درست خود قرار می‌دهیم. با این روش حداکثر $O(n^2)$ جابجایی خواهیم داشت. چون در مرحله iام حداکثر $n-i+O(1)$ جابجایی داریم.

(ب) مجاز هستیم از یک برج هانوی دیگر و به تعداد k دیسک روی زمین قرار دهیم.

راه حل اینجا تعمیمی از راه حل بالاست. چون اینجا مجاز هستیم تا k دیسک روی زمین قرار دهیم، در مرحله اول k دیسک بالای S_1 را روی زمین قرار می‌دهیم و مانند قبل بقیه محتوای S_1 را به S_2 منتقل می‌کنیم. سپس دیسکهای روی زمین را به ترتیب درست داخل S_1 قرار می‌دهیم و دیگر کاری به آنها نداریم. محتوای برج S_2 را به برج S_1 برمی‌گردانیم. در پایان این مرحله k دیسک به جای درست خود متنقل شده اند و $n-k$ دیسک باقی مانده‌اند که وارون شوند. همین روند را برای $n-k$ عنصر بالای برج S_1 تکرار می‌کنیم و هر دفعه k دیسک جدید را در جای درست خود قرار می‌دهیم. شکل صفحه بعد الگوریتم را برای $k=3$ و $n=8$ نشان می‌دهد.

مرحله اول شامل $O(n)$ جابجایی است. مرحله دوم شامل $O(n-k)$ جابجایی. مرحله سوم شامل $O(n-2k)$ جابجایی تا آخر. لذا تعداد کل جابجایی ها حداکثر

$$\sum_{i=0}^{n/k} n-ik=\Theta(\frac{n^2}{k})$$


S1 a b c d e f g h
S2
S1 d e f g h
S2
a b c
S1
S2 h g f e d
a b c
S1 c b a
S2 h g f e d
S1 d e f g h c b a
S2
S1 g h c b a
S2
d e f
S1 c b a
S2 h g
d e f
S1 f e d c b a
S2 h g
S1 f e d c b a
S2
h g
S1 h g f e d c b a
S2

(ج) مجاز هستیم از دو برج هانوی دیگر و به تعداد $O(1)$ دیسک روی زمین قرار دهیم.

دیسکها را یکی یکی از بالای برج $S1$ برمیداریم و در برج $S2$ درج می‌کنیم. محتوای برج $S2$ وارون برج $S1$ خواهد شد. حال دیسکها را یکی یکی از برج $S2$ به $S3$ منتقل میکنیم. در نهایت، عناصر را از $S3$ به $S1$ منتقل می‌کنیم. برج $S1$ وارون شده است. زمان اجرا $O(n)$ است.

۲. گزاره‌های زیر را در مورد درخت باینری ثابت کنید. n_E تعداد برگهای درخت و n_I تعداد رئوس غیر برگ است.

1. $h+1 \leq n \leq 2^{h+1}-1$
2. $1 \leq n_E \leq 2^h$
3. $h \leq n_I \leq 2^h - 1$
4. $\log(n+1)-1 \leq h \leq n-1$

توجه: در اینجا ارتفاع برگ صفر محسوب می‌شود.

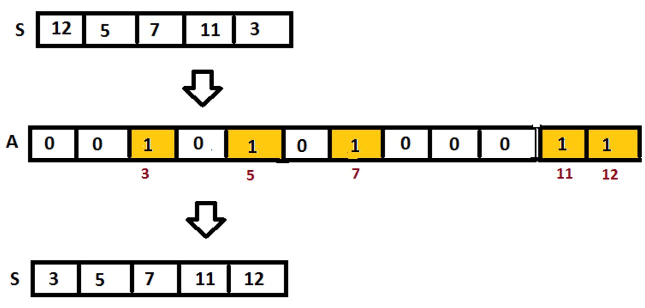
- (1) روشن است برای داشتن ارتفاع h تعداد رئوس باید حداقل $h+1$ باشد. از طرفی دیگر، درختی با ارتفاع h حداکثر $2^{h+1}-1$ راس دارد (ماکزیمم وقتی است که درخت کاملا پر باشد یعنی درخت باینری کامل باشد.)
- (2) واضح است هر درخت باینری حداقل یک برگ دارد. حداکثر تعداد برگها هم وقتی است که درخت باینری کامل باشد. چون اگر درخت کامل نباشد، می‌توان بدون افزایش ارتفاع، با اضافه کردن رئوس تعداد برگها را افزایش داد.
- (3) چون $n_I = n - n_E$، از (1) و (2) نتیجه می‌شود که $h \leq n_I \leq 2^h - 1$. کافی است دو نامساوی (1) و (2) را از هم کم کنید.
- (4) از نامساوی (1) نتیجه می‌شود.

۳. در یک درخت دودویی کامل، هر راس یا فرزند ندارد یا دو فرزند دارد. نشان دهید تعداد رئوس در یک درخت باینری کامل فرد است.

پروسه‌ای را در نظر بگیرید که درخت مورد نظر را ایجاد کرده است. درخت هر بار با اضافه کردن دو فرزند به یک راس گسترش می‌یابد. پس هر بار 2 راس جدید به درخت اضافه کرده‌ایم. فرض کنید تعداد دفعاتی که این کار را انجام داده‌ایم، k باشد. با در نظر گرفتن ریشه تعداد رئوس درخت برابر با $2k+1$ خواهد بود که یک عدد فرد است.

۴. توضیح دهید که n عدد صحیح که از بازه $[1, 100n]$ هستند را چگونه می‌توان در زمان $O(n)$ مرتب کرد؟ چرا این با کران پایین $\Omega(n \log n)$ برای مرتب سازی در تناقض نیست؟

چون اعداد صحیح هستند و در بازه $[1, 100n]$ قرار دارند، می‌توانیم یک آرایه خالی A با مقدار اولیه صفر به اندازه $100n$ (با اندیسهای 1 تا $100n$) ایجاد کنیم. حال لیست ورودی S را می‌خوانیم و $S[i]$ را در محل $A[S[i]] = 1$ درج می‌کنیم. پس درج عناصر در لیست A، لیست A را می‌خوانیم و عناصر غیر صفر آن را به ترتیب در لیست S درج می‌کنیم. لیست S مرتب شده است. همه این اعمال در زمان $O(n)$ انجام شده است. دقت کنید که اینجا لیست S را با روش مقایسه عناصر مرتب نکردیم و لذا کران پایین $\Omega(n \log n)$ برای مرتب سازی اینجا مطرح نیست.



۵. فرض کنید در هرم بیشینه فرزند سمت چپ دختر و فرزند سمت راست پسر باشد. در نمایش هرم با آرایه، عنصری که در اندیس iام ذخیره شده، پسر دخترش در چه اندیسی ذخیره شده؟

$4i+1$

اگر ریشه دختر باشد، در یک هرم بیشینه با n راس حداقل چند دختر وجود دارد؟

هر پسر حتما یک خواهر دارد ولی ممکن است دختری باشد که برادر ندارد. لذا بدون احتساب ریشه، تعداد دخترها $\lceil (n-1)/2 \rceil$ است. چون ریشه را هم دختر حساب کردیم پس تعداد دخترها دقیقا $\lceil (n-1)/2 \rceil + 1$ است.

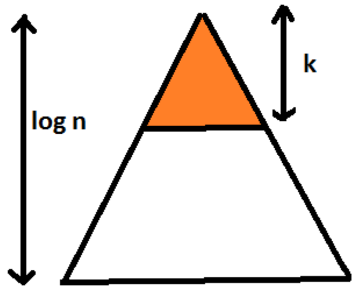
۶. کوچکترین عنصر در یک هرم بیشینه را در چه زمانی می‌توان پیدا کرد؟ چرا؟

کوچکترین عنصر در هرم بیشینه می‌تواند در هر برگی از درخت قرار گرفته باشد. تعداد برگها در یک هرم با n عنصر حداقل $n/2$ است لذا باید حداقل $n/2$ مکان مختلف را برای پیدا کردن عنصر کمینه چک کنیم و لذا زمان جستجو $O(n)$ است.

k امین بزرگترین عنصر را در چه زمانی می‌توان پیدا کرد؟

می‌توانیم $k-1$ بار عنصر واقع در ریشه را حذف کنیم و سپس ریشه درخت حاصل را به عنوان k امین بزرگترین عنصر گزارش کنیم. زمان اجرای این ایده $O(k \log n)$ است چون هر عمل حذف زمانش $O(\log n)$ است.

یک ایده بهتر می‌توان بدین صورت باشد. با توجه به اینکه k امین بزرگترین عنصر یکی از رئوس واقع در k سطح بالای درخت است، لذا موقع حذف و بازسازی هرم، کافی است که حداکثر به تعداد k سطح از ریشه پایین برویم و جابجایی ها را تا آن سطح انجام دهیم. چون $k-1$ بار حذف می‌کنیم، زمان اجرای این ایده $O(k^2)$ خواهد بود. دقت کنید در این حالت برای $k=O(1)$ زمان اجرا $O(1)$ خواهد بود که از ایده قبلی بهتر است. اما اگر k بیشتر از $\log n$ باشد، ایده قبلی سریعتر است.

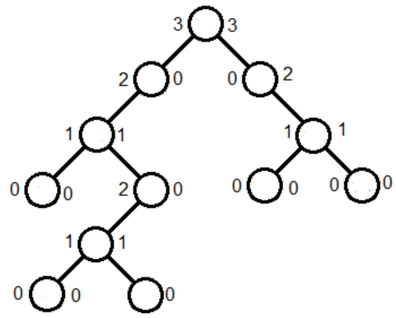


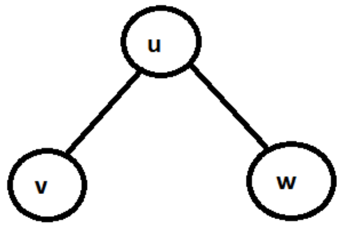
حتی یک ایده بهتر از اینها هم وجود دارد. فرض کنید H هرم ورودی باشد. این ایده بر این مشاهده استوار است که k امین بزرگترین عنصر در هرم بیشینه H، فرزند یکی از $k-1$ عنصر بزرگ هرم است. برای پیدا کردن سریع این فرزند، می‌توانیم از یک هرم بیشینه کمکی استفاده کنیم. هرم H' در ابتدا خالی است. بزرگترین عنصر هرم H را استخراج کرده و در H' وارد می‌کنیم. حال هر بار ریشه H' را حذف کرده و دو فرزند آن در H را در H' درج می‌کنیم. این کار را $k-1$ بار تکرار می‌کنیم. بعد از $k-1$ بار، ریشه هرم H' عنصری است که دنبال آن می‌گشتیم. چون به تعداد $k-1$ بار از هرم H' حذف می‌کنیم و تعداد عناصر هرم H' حداکثر $2k$ است لذا زمان اجرا $O(k \log k)$ خواهد بود.

۷. درخت دودویی T شامل n راس داده شده است. شیب چپ راس x برابر با تعداد قدمهایی است که از x میتوانیم به سمت پایین برویم به شرطی که هر بار به سمت چپ برویم. شیب راست x هم به همین منوال قابل تعریف است. یعنی تعداد دفعاتی که با شروع از راس x میتوانیم به سمت راست برویم. در شکل زیر، شیب چپ و شیب راست هر راس در کنار آن نوشته شده است.

(آ) الگوریتمی طراحی کنید که شیب چپ و راست هر راس درخت را محاسبه کند. زمان اجرای الگوریتم شما چقدر است؟

ایده کلی راه حل: فرض کنید $\ell(u)$ شیب چپ راس u و $r(u)$ شیب راست آن باشد. با توجه تعریف داده از شیب چپ و راست، اگر u فرزند چپش v باشد داریم $\ell(u) = \ell(v) + 1$ و همینطور اگر w فرزند راست u باشد داریم





$r(u) = r(w) + 1$. لذا شیب چپ و راست رئوس درخت را می‌توان با یک پروسه پایین به بالا (مانند محاسبه ارتفاع رئوس درخت) محاسبه کرد. زمان اجرای الگوریتم $O(n)$ است.

(ب) میخواهیم ساختار داده ای طراحی و پیاده سازی کنیم که سه عمل اصلی داشته باشد.

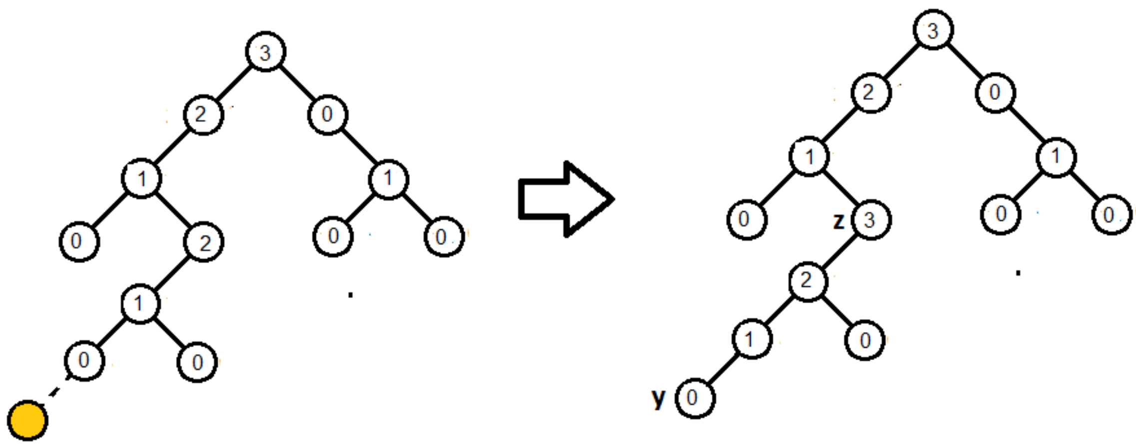
i. اضافه کردن ریشه در صورتی که درخت تهی باشد.

ii. یک راس با بیشترین شیب چپ یا راست را گزارش کند. همراه با مقدار شیب

iii. اضافه کردن یک برگ به درخت. فرض بر این است، که آدرس راس x را میدانیم و میخواهیم یک فرزند چپ یا راست به x اضافه کنیم. در صورتی که فرزند قبلا موجود باشد، پیام خطا چاپ شود.

(ج) در راه حل شما زمان اجرای هر یک از اعمال بالا چقدر است؟

ایده کلی یک راه حل. برای سادگی فرض کنید فقط شیب چپ را می‌خواهیم حساب کنیم. محاسبه شیب راست به طریق مشابه انجام می‌شود. ابتدا توجه کنید که اضافه کردن یک فرزند چپ می‌تواند شیب چپ چندین راس در درخت را تغییر دهد. در شکل زیر یک مثال نشان داده شده است. داخل هر راس شیب چپ آن نوشته شده است. اضافه شدن یک فرزند چپ باعث تغییر شیب چند عنصر بالای آن شده است.



برای هر راس u، تعریف می‌کنیم $B(u)$ پایین ترین جد u که فرزند راست پدرش باشد. اگر u فرزند راست پدرش باشد قرار می‌دهیم $B(u) = u$. برای مثال در شکل بالا داریم $B(y) = z$ و $B(z) = z$. دقت کنید با این تعریف

هر وقت برگ x به درخت اضافه می‌شود، به شیب چپ همه اجداد x از پایین تا $B(x)$ یک واحد اضافه می‌شود (شام خود x نمی‌شود.)

در حین اجرا، برای هر راس u، یک اشاره‌گر به $B(u)$ نگه می‌داریم. وقتی که برگ x اضافه می‌شود، $B(x)$ را می‌توان از روی وضعیت پدرش p و $B(p)$ آن بدست آورد.

برای استخراج راس با بیشترین شیب چپ، می‌توانیم از یک هرم بیشینه استفاده کنیم. هر برگ x که به درخت اضافه می‌شود به عنوان یک عنصر جدید به هرم اضافه می‌شود که مقدار اولیه آن صفر است. اضافه شدن برگ x باعث می‌شود که شیب راس $B(x)$ هم یک واحد اضافه شود. شیب راسهای دیگر هم اضافه می‌شوند ولی آنها اهمیتی ندارند چون هیچ وقت بیشترین از آن آنها نیست. لذا در هرم بیشینه یک عمل افزایش کلید برای $B(x)$ داریم. به این ترتیب، هر زمان عنصر با بیشترین شیب چپ را می‌توان از ریشه هرم استخراج کرد که زمان آن $O(1)$ است. زمان اضافه کردن برگ با توجه به بروزرسانی هرم، $O(\log n)$ خواهد بود.