ساختمان داده‌ها و الگوریتم‌ها

نیم‌سال دوم ۹۷ـ۹۸

دانشکده‌ی مهندسی کامپیوتر

گردآورندگان: کیارش بنی هاشم، حمیدرضا هدایتی

پاسخ تمرین دوم — تحلیل سرشکن، پشته، صف، پیمایش درخت

تحلیل سرشکن

مسئله‌ی ۱. چندپشتک

داده ساختار چندپشتک، داده ساختاری است متشکل از دنباله ای نامتناهی از پشته ها مانند $S_0, S_1, ...$ به طوری که ظرفیت پشته S_i، 2^i می باشد.
اگر کاربر بخواهد عنصری در پشته پر S_i درج کند، باید ابتدا تمامی عناصر S_i را حذف کند و در S_{i+1} درج کند تا S_i خالی شود، سپس عنصر مورد نظر را در S_i درج کند. (اگر S_{i+1} پر باشد، ابتدا به صورت بازگشتی، عناصر آن را در S_{i+2} درج می کنیم).
فرض کنید که عملیات درج ($push$) و حذف (pop) در هر پشته از مرتبه $O(1)$ باشد.

الف

در بدترین حالت، چقدر طول می کشد که یک عنصر جدید، در یک چندپشتکی که n عنصر دارد، درج شود؟

ب

ثابت کنید که هزینه سرشکن عملیات درج در یک چند پشتک از مرتبه $O(log(n))$ است. (در صورتی که n بیشینه تعداد عناصر موجود در چند پشتک باشد)

حل.

الف

از آن جایی که ممکن است پشته با ظرفیت $2^{\lfloor lg(n) \rfloor}$ پر باشد، و تعداد اعضای آن از مرتبه $\theta(n)$ است، حداکثر $\theta(n)$ طول می کشد.

ب

روش انبوهه:
فرض کنید که کاربر، n عنصر درج کرده است. در این صورت، پشته اول، حداکثر n بار، پشته دوم، حداکثر $\frac{n}{۲}$ بار و ... پشته i ام حداکثر $\frac{n}{۲^i}$ بار خالی شده است زیرا با هر بار خالی شدن پشته i ام، ۲^i عنصر، به پشته های بعد از i انتقال می یابند و دیگر نمی توانند وارد پشته i شوند. از آنجایی که هزینه خالی کردن پشته i ام، ۲^i است، هزینه کل، کمتر مساوی است از

$$\sum_{i=۱}^{lg(n)} \frac{n}{۲^i} \times ۲^i = O(nlog(n))$$

روش سکه:
هر بار که یک عنصر را درج می کنید، $log(n)$ سکه روی آن بگذارید. این سکه ها قرار است هزینه جابجایی این عنصر را بدهند.
از آن جایی که پشته $log(n)+۱$ به بعد، هرگز پر نمی شوند، هر عنصر، حداکثر $O(log(n))$ بار جابجا می شود پس سکه های ما می توانند هزینه تمامی عملیات های جابجایی را بدهند
روش پتانسیل:
برای سادگی تحلیل، کمی فرض مساله را تغییر می دهیم، به جای این که فرض کنیم که اگر یک پشته پر است، درج رخ نمی دهد، فرض می کنیم که درج رخ می دهد (مثلا در یک حافظه فرضی موقت) و بلافاصله پس از درج، جابجایی صورت می گیرد.
اگر تعداد عناصر موجود در پشته i ام را با $k(i)$ نشان دهیم، قرار دهید

$$\Phi = \sum_{i<\lceil log(n) \rceil} (\lceil log(n) \rceil - i)k(i)$$

در واقع مثل این است که به هر عضو، پتانسیل $\lceil log(n) \rceil$ منهای شماره پشته ای که در آن قرار دارد را نسبت داده ایم. در این صورت، از آن جایی که در هر مرحله، هنگام درج، تابع پتانسیل، حداکثر به اندازه $\lceil log(n) \rceil$ افزایش می یابد، هزینه از مرتبه $\theta(log(n))$ است. از طرفی، دقت کنید که اگر پشته i بیشتر از پر شود، به راحتی می توانید با استقرا ثابت کنید تعداد اعضای آن، حداکثر برابر است با $۲^{i+۱}$ و در نتیجه از مرتبه ۲^i است . پس مقدار کاهش تابع پتانسیل، برابر است با

$$۲^i \times ((\lceil log(n) \rceil - i) - (\lceil log(n) \rceil - (i+۱))) = ۲^i$$

در نتیجه، کاهش پتانسیل، می تواند هزینه جابجایی ها را بدهد. ◁

مسئله‌ی ۲. شمارنده دودویی

یک شمارنده n بیتی دودویی در نظر بگیرید. که هزینه تعویض بیت i ام آن i است. مثلا اگر شمارنده ۱۰۰۱۱۱ باشد هزینه افزایش آن به ۱۰۱۰۰۰ (عدد بعدی) برابر است با

$$۱+۲+۳+۴=۱۰$$

الف

ثابت کنید که هزینه سرشکن افزایش یک واحد این شمارنده، $O(۱)$ است

ب

فرض کنید هزینه تعویض بیت i ام به جای i، ۲^i باشد، و شمارنده را از صفر تا عدد k افزایش دهیم. ثابت کنید هزینه سرشکن هر افزایش، $O(log(k))$ خواهد بود.

حل.

الف

روش انبوهه:
اگر تا عدد k افزایش دهیم، بیت i ام، $\frac{k}{۲^i}$ بار تغییر می کند، پس هزینه برابر است با

$$\sum_{i=۱}^{log(k)} i \times \frac{k}{۲^i} = O(k)$$

روش سکه:
ثابت می کنیم در حالتی که فقط تغییر ۰ به ۱ هزینه i دارد و تغییر ۱ به صفر هزینه ای ندارد، افزایش ۱ واحد، $O(۱)$ است. دقت کنید که اثبات این ادعا، حکم را نتیجه می دهد زیرا اگر این ادعا ثابت شود، در حالتی که افزایش بیت i ام از ۰ به ۱، $۲i$ هزینه داشته باشد نیز صادق است زیرا هزینه ها صرفا ۲ برابر می شود. از طرفی چون هر تغییر از ۱ به ۰، متناظر با یک تغییر قبلی از ۰ به ۱ است، یعنی می توان هزینه تغییر از ۱ به صفر را در هنگام تغییر از ۰ به ۱ پرداخت کرد، حکم نتیجه می شود.
برای اثبات این ادعا، دقت کنید که هنگام افزایش بیت i، بیت های ۱ تا $i-۱$، برابر با ۱ هستند. با توجه به این نکته، طوری سکه قرار دهید که برای افزایش بیت i ام، از سکه هایی که روی بیت های ۱ تا $i-۱$ قرار دارند استفاده شود و سپس روی بیت i ام، یک سکه جدید قرار گیرد (این سکه برای استفاده بیت ها بعدی در آینده می باشد)
روش پتانسیل:

اگر A مجموعه همه i هایی باشد که بیت i ام، ۱ است، تعریف کنید

$$\Phi = \sum_{i \in A} (i + ۲)$$

در این صورت، اگر s بیت اول، ۱ باشند و بیت بعدی ۰ باشد، پس از افزایش ۱ واحد، کاهش

پتانسیل برابر خواهد بود با

$$\sum_{i=۱}^{s}(i+۲)-(s+۱+۲)=(\sum_{i=۱}^{s}i)+۲s-s-۳=\sum_{i=۱}^{s+۱}i-۳$$

در نتیجه با پرداخت هزینه ۳ که از مرتبه $O(۱)$ است و استفاده از کاهش پتانسیل فوق، می توان افزایش یک واحد را انجام داد.
معادلا می توانید طبق توضیحات بخش قبل، مساله را در حالتی که فقط افزایش بیت ها هزینه دارد حل کنید و قرار دهید

$$\Phi=|A|$$

ب

روش انبوهه:
مشابه قبل

$$\sum_{i=۱}^{log(k)}۲^i\times\frac{k}{۲^i}=O(klog(k))$$

روش سکه: قرار دهید $z=log(k)$، فقط z بیت اول تغییر می کنند. هر مرحله، روی هر z بیت یک سکه قرار دهید. از آن جایی که بین هر دو تغییر بیت i ام، باید $\theta(۲^i)$ افزایش واحد داشته باشیم، می توان از سکه هایی که روی آن جمع شده اند استفاده کرد.
روش پتانسیل:
مشابه قبل، قرار دهید $z=log(k)$.
تابع زیر را در نظر بگیرید

$$\phi=\sum_{i\in A}(z-i)۲^i$$

نشان دهید عبارت زیر برقرار است.

$$\sum_{i=۱}^{n}i۲^i=(n-۱)۲^{n+۱}+۲$$

پس که اگر s بیت اول، ۱ باشند و بیت بعدی صفر، با افزایش یک بیت، مقدار افزایش پتانسیل برابر است با

$$(z-s-۱)۲^{s+۱}-\sum_{i=۱}^{s}(z-i)۲^i$$

از طرفی

$$\sum_{i=۱}^{s}(z-i)۲^i=z(\sum_{i=۱}^{s}۲^i)-(s-۱)s^{s+۱}-۲=z(۲^{s+۱}-۱)-(s-۱)۲^{s+۱}-۲$$

پس افزایش پتانسیل برابر است با

$$z - \theta(۲^s)$$

از آنجایی که هزینه تغییرات، $\theta(۲^s)$ است (مجموع هزینه تغییر بیت های ۱ تا s، یک سری هندسی است که به راحتی قابل محاسبه است)، با هزینه سرشکن z، می توان این هزینه را پرداخت.

▷

مسئله‌ی ۳. آرایه نمایی

قصد داریم داده ساختار پشته را پیاده سازی کنیم. این داده ساختار باید قابلیت های درج و حذف داشته باشد. همانطور که می دانید، یک راه برای پیاده سازی پشته این است که آرایه ای از اعداد در نظر بگیریم و تا زمانی که آرایه پر نشده، اعداد را در این آرایه درج کنیم و هنگامی که پر شد، آرایه ای دیگر با ۲ برابر اندازه در نظر گرفته و اعداد را به آن منتقل کنیم.
یک مشکلی که این روش دارد، این است که اگر تعداد زیادی عنصر درج شده، و سپس همه حذف شوند، آرایه بزرگ می ماند. برای رفع این مشکل قصد داریم هنگامی که تعداد عناصر موجود در آرایه از نصف بیشینه ظرفیت آن کمتر شد، آرایه ای با نصف اندازه در نظر گرفته، و تمامی اعضا را به آن انتقال دهیم.

الف

دنباله ای از درج و حذف ها پیشنهاد دهید که برای آن، هزینه سرشکن درج و یا هزینه سرشکن حذف، $O(۱)$ نباشد.

ب

ثابت کنید که اگر به جای این که هنگامی که آرایه تا نصف پر شد، آرایه ای با نصف اندازه در نظر بگیریم، هنگامی که آرایه تا یک چهارم پر شد، آرایه با نصف اندازه در نظر بگیریم، هزینه عمل درج و عمل حذف، $O(۱)$ است.

حل.

الف

۱ – ۲^i عنصر در آرایه درج کنید. سپس عملیات زیر را مرتب تکرار کنید

- درج ۲ عنصر
- حذف ۲ عنصر

چون هر گاه اندازه از ۲^i بیشتر شد، دو برابر می شود، با درج ۲ عنصر، تمامی اعضای آرایه باید جابجا شوند. با حذف ۲ عنصر، مجددا چون تعداد عناصر از نصف کمتر است، باید تمامی اعضا جابجا شوند.

ب

پس از هر عملیات جابجایی (چه به آرایه کوچتر و چه به بزرگتر)، آرایه جدید، تا نصف پر است. برای این که این تعادل به هم بخورد، باید $\Omega(n)$ عملیات انجام شود زیرا باید حداقل $\frac{n}{۴}$ عملیات انجام شود تا آرایه بیش از حد کم شود و یا حداقل $\frac{n}{۲}$ عملیات درج انجام شود تا آرایه بیش از حد پر شود. از آن جایی که هزینه جابجایی، $\theta(n)$ می باشد، می توان این هزینه را بین این عملیات ها تقسیم کرد. مثلا اگر هزینه جابجایی برای n عنصر، دقیقا n باشد، اگر هنگام عملیات درج، ۲ سکه و هنگام عملیات حذف، ۱ سکه ذخیره کنیم، می توان با استفاده از سکه های جمع شده، عملیات جابجایی را انجام داد. ◁

پشته و صف

مسئله‌ی ۴. صد و نه

اعداد طبیعی n و z، به همراه یک دنباله n تایی a از اعداد حقیقی داده شده است. می خواهیم بدانیم که آیا دو اندیس i و j یافت می شوند که

$$|i-j| < z, a[i] - a[j] > ۱۰۹$$

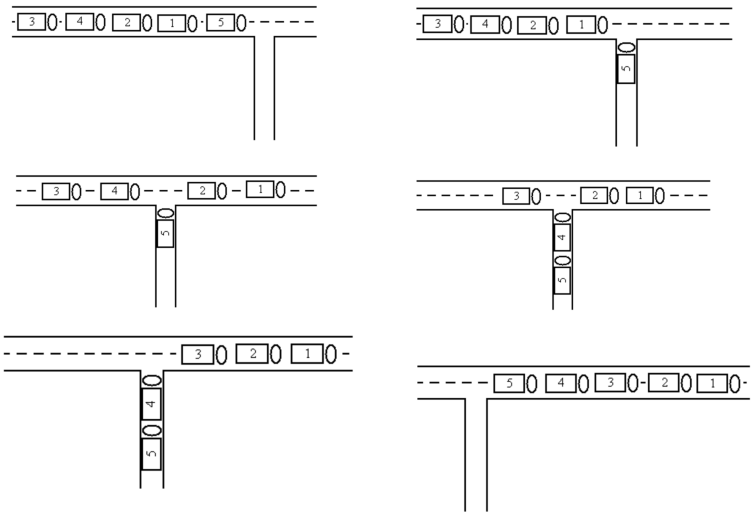
الگوریتمی از $O(n)$ برای حل مساله بدهید.

حل. برای این کار، ابتدا مینیمم عدد در هر بازه $k = ۲z + ۱$ تایی را پیدا می کنیم. سپس، یک بار لیست اعداد را پیمایش می کنیم و برای هر عدد، آن را با مینیمم اعداد در بازه k تایی که مرکز آن است مقایسه می کنیم.
برای یافتن مینیمم، از داده ساختار صف استفاده کنید و سعی کنید، در هر مرحله مانند i، اعضای صف اعدادی باشند مانند $a[j]$ که اولا $i - j < k$ و دوما در بازه (j, i) مینیمم باشند. ◁

مسئله‌ی ۵. قطار

برای تولید جایگشت های اعداد ۱ تا n، از ماشین artemis که به شکل زیر کار می کند استفاده می کنیم.
ماشین یک ریل ورودی، یک ریل خروجی و یک ریل برای ذخیره اعداد میانی دارد. هر عدد، از سمت چپ وارد ماشین شده و می تواند مستقیم از سمت راست خارج شود و یا وارد ریل میانی شود. ریل میانی مانند یک پشته عمل می کند بنابراین تنها عنصر بالایی آن می تواند خارج شود.

ماشین، دنباله ی اعداد ۱ تا n را ورودی گرفته و جایگشتی از آن را خروجی می دهد. برای مثال اگر ورودی جایگشت ۵, ۱, ۲, ۴, ۳ باشد، می توان برای تولید جایگشت ۵, ۴, ۳, ۲, ۱ به شکل زیر عمل کرد



در هر دو بخش سوال، فرض کنید که ورودی، جایگشت $1, 2, ...n$ ۱, ۲, ... می باشد

الف

تعداد جایگشت های قابل تولید با ماشین فوق را به دست آورید.

ب

الگوریتمی از مرتبه زمانی خطی ارایه دهید که یک جایگشت از اعداد ۱ تا n را ورودی بگیرد و مشخص کند که آیا این جایگشت، توسط ماشین فوق، قابل تولید هست یا نه؟

حل.

الف

دقت کنید که این قابلیت که هر ماشین می تواند مستقیم و بدون ورود به ریل میانی وارد ریل خروجی شود، تاثیری در جایگشت های قابل تولید ندارد. می توان به جای این که یه عدد را مستقیم وارد ریل خروجی کرد، ابتدا وارد ریل میانی کرد و سپس در مرحله بعدی، همان عنصر را از ریل میانی خارج کرد. پس کافیست مساله ساده تر زیر را حل کنیم.
در هر مرحله، یا یک عنصر از ریل ورودی وارد ریل میانی می شود و یا یک عنصر از ریل میانی خارج می شود (اگر ریل میانی خالی نباشد).

مشابه مساله تعداد پارانتز ها می باشد و جواب آن، عدد کاتالان است. برای اثبات، کافیست اولین باری که ریل میانی خالی می شود را در نظر گرفته و نتیجه بگیرید که رابطه بازگشتی جواب مساله، همان رابطه بازگشتی کاتالان است.

ب

ماشین را شبیه سازی کنید. برای چاپ کردن اولین عنصر جایگشت، باید آن قدر عنصر وارد ریل میانی شود تا به عنصر مورد نظر برسیم. سپس عنصر مورد نظر باید وارد ریل خروجی شود. حال، سعی می کنیم عنصر بعدی جایگشت را تولید کنیم، اگر این عنصر در ریل ورودی باشد و یا عنصر رویی ریل میانی باشد، آن را وارد ریل خروجی کرده و ادامه می دهیم، وگرنه یعنی در حالتی که این عنصر در ریل میانی است ولی عضو رویی نیست، نمی توان جایگشت مورد نظر را تولید کرد.(دقت کنید که عملکرد ماشین برای تولید یک جایگشت عملا یکتاست، برای این که کاملا یکتا شود، می توان از ساده سازی بخش الف استفاده کرد) ◁

مسئله‌ی ۶. پارانتز و براکت

رشته ای متشکل از تعدادی کاراکتر پارانتز ’)’، ’(’ و تعدادی براکت ’]’، ’[’ در اختیار داریم. الگوریتمی از $O(n)$ ارایه دهید که تشخیص دهد که آیا پارانتز و براکت گذاری در این رشته صحیص است یا نه. به عنوان مثال، رشته زیر صحیح نیست

([)]

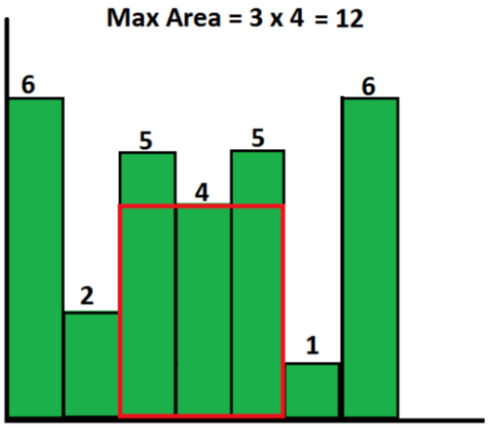
ولی رشته زیر صحیح است

([])((([])())

حل. هر گاه به یک پارانتز یا براکت باز رسیدید، در استک پوش کنید و هر وقت به یکی که بسته است رسیدید، سعی کنید متناظر آن را پاپ کنید. اگر نتوانستید، رشته مشکل دارد.
همچنین دقت کنید که در انتها باید استک خالی باشد.
◁

مسئله‌ی ۷. بیشینه مساحت

دنبال ای از اعداد $a_1, ...a_n$ به شما داده شده است.
همانند شکل زیر، مستطیل هایی با عرض یکسان و ارتفاع a_i، را در کنار هم قرار داده ایم.
الگوریتمی از مرتبه زمانی خطی ارایه دهید که بزرگترین زیرمستطیل را از نظر مساحت بیابد



حل. مستطیلی که بیشترین مساحت را در نظر بگیرید و فرض کنید که ارتفاع آن h باشد و عرض آن بازه x تا y باشد.
از آن جایی که نمی توان مستطیلی با مساحت بیشتر در نظر گرفت، اندیس z یافت می شود که بین x, y باشد و $h = a_z$. همچنین از آنجایی که نمی توان مستطیل را از راست و یا چپ گسترش داد، $a_{x-1}, a_{y+1} < h$
برای حل مساله، با استفاده از پشته و با یک بار پیمایش آرایه a، به ازای هر اندیس i، راست ترین مستطیلی که ارتفاعش از a_i بیشتر مساوی است را بیابید. مشابها، با یک پیمایش دیگر، چپ ترین مستطیلی که ارتفاعش از i بیشتر مساوی است را بیابید و این دو مقدار را برای هر اندیس، ذخیره کنید. مثلا در آرایه های r و l.
برای حل مساله اصلی، ماکسیمم عبارت زیر را محاسبه کنید

$$(r[i] - l[i]) * a_i$$

طبق استدلال های بالا، در واقع بیشینه مساحت، مقدار فوق برای $i = z$ می باشد. از طرفی چون به وضوح می توان مستطیل با مساحت $(r[i] - l[i]) * a_i$ را در نظر گرفت، الگوریتم درست است.
▷

لیست پیوندی

مسئله‌ی ۸. آینه پیوندی

فرض کنید که یک رشته به طول n را در یک لیست پیوندی یک طرفه ذخیره شده و می خواهیم بدون تغییری در لیست، بررسی کنیم که آیا این رشته، آینه ای است یا نه

الف

با استفاده از حافظه اضافی $O(۱)$، الگوریتمی از مرتبه زمانی $O(n^۲)$ برای این کار ارایه دهید

ب

با استفاده از حافظه اضافی $O(log(n))$، الگوریتمی از مرتبه زمانی $O(nlog(n))$ برای این کار ارایه دهید

حل.

الف

اگر لیست پیوند ما دو طرفه بود، کافی بود که یک اشاره گر به ابتدا داشته باشیم، یک اشاره گر به انتها، و در هر مرحله، دو کاراکتر را مقایسه کنیم، سپس قرار دهیم

$$p۱ = next[p۱], p۲ = prev[p۲]$$

از آن جایی که لیست یک طرفه است، برای محاسبه $prev[p۲]$، از $p۱$ شروع به حرکت به جلو می کنیم تا این که به عضو k از لیست برسیم به طوری که $next[k] = p۲$ و قرار می دهیم $p۲ = k$.

ب

در الگوریتم بخش قبل، مشکلی که باعث می شد مرتبه زمانی الگوریتم بالا باشد این بود که زمان بسیار زیادی طول می کشد که $prev[p۲]$ یافت شود زیرا ممکن است که فاصله $p۱$ تا $p۲$ زیاد باشد و مقدار زمانی که طول می کشد تا به $p۲$ برسد.
برای رفع مشکل، از $log(n)$ اشاره گر استفاده می کنیم. به طور دقیق تر اگر $p۲$ به k امین عنصر لیست اشاره کند، یک اشاره گر به k امین عنصر، یک اشاره گر به بزرگترین عدد زوج کمتر مساوی k، یک اشاره گر به بزرگترین عدد مضرب ۴ کمتر از k، ... یک اشاره گر به بزرگترین مضرب $۲^{log(n)}$ کمتر از k نگه می داریم.
در این صورت، برای یک واحد کاهش k، به اولین بیتی از k که از ۱ به ۰ تبدیل می شود نگاه می کنیم، مثلا فرض کنید بیت i ام باشد. کافیست که اشاره گر متناظر با بیت i ام را در نظر بگیریم و از آن اشاره گر شروع به حرکت به سمت k کنیم.
برای تحلیل زمانی الگوریتم، می توانید از بخش ب سوال قبلی استفاده کنید.

▷

مسئله‌ی ۹. دور کم حافظه

الگوریتمی ارایه دهید که بدون تغییر ساختار یک لیست پیوندی و با استفاده از حافظه $O(۱)$، در

زمان خطی تشخیص دهد که این لیست دور دارد یا خیر؟

حل.
دو اشاره گر به ابتدای لیست نگه دارید. در هر مرحله، با استفاده از تابع next، اولی را یک واحد افزایش دهید و دومی را ۲ واحد، سپس این دو اشاره گر را مقایسه کنید، اگر یکسان بودند دور وجود دارد.
برای اثبات درستی، دقت کنید که اگر دور نباشد، هرگز این دو اشاره گر یکی نمی شوند و در نهایت، در زمان خطی، یکی از آن ها به ته لیست می رسد. (اول، اشاره گر دوم می رسد)
اگر دور وجود داشته باشد، پس از زمان خطی، هر دو عنصر وارد دور می شوند. هنگامی که دو عنصر درون دور قرار گرفتند، چون در هر مرحله، یکی با سرعت ۲ و دیگری با سرعت ۱ حرکت می کند، مثل این است که یکی ثابت است و دیگری با سرعت 1- حرکت می کند. پس در زمان خطی به هم می رسند.
▷

پیمایش درخت

مسئله‌ی ۱۰. مسیر سنگین

در یک درخت دودویی (لزوما درخت جست و جو نیست) منظور از وزن یک مسیر، مجموع اعداد روی راس های آن مسیر است.
الگوریتمی با زمان خطی ارایه دهید، که سنگین ترین مسیر را بیابد

حل.

درخت را به صورت پس ترتیب پیمایش کنید و به صورت بازگشتی، برای هر عنصر دو مقدار زیر را به دست آورید

- سنگین ترین مسیر زیر درخت
- سنگین ترین مسیر زیردرخت که یک انتهای آن ریشه است

▷

مسئله‌ی ۱۱. درخت کم حافظه

الگوریتمی غیر بازگشتی ارایه دهید که با $O(1)$ حافظه اضافی، یک درخت دودویی را پیمایش پس ترتیب کند و همه اعضای آن را چاپ کند.

حل.
اگر حافظه بیشتری داشتیم، می توانستیم الگوریتم زیر را انجام دهیم
برای هر عنصر، ۳ مقدار boolean نگه داریم که به ترتیب نشان دهند که آیا زیر درخت چپ عنصر

پیمایش شده یا نه، زیر درخت راست آن پیمایش شده یا نه، و این که آیا خود عنصر چاپ شده یا نه. در این صورت اگر اشاره گر ما روی یک عنصر باشد، با توجه به این ۳ مقدار، می توان تصمیم گرفت که در مرحله بعدی باید به کجا برود. مثلا اگر زیردرخت چپ پیمایش شده ولی راست نشده، به راست برود یا اگر هر ۳ مقدار boolean، درست بودند، به عنصر پدر خود برود. متاسفانه به این حافظه دسترسی نداریم، اما دقت کنید که با توجه به حرکت های قبلی اشاره گر ما، می توان تصمیم درست را گرفت. به طور دقیق تر، اگر اشاره گر ما روی A باشد و در مرحله قبلی روی پدر A بوده باشد، در این صورت، تازه وارد زیردرخت A شده و باید به زیردرخت چپ برود. اگر در مرحله قبلی روی فرزند چپ A بوده باشد، باید به فرزند راست برود و در نهایت اگر روی فرزند راست بوده باشد، باید A چاپ شود و به پدر A برود.
پس کافیست مقدار سابق A را نگه داریم و در نتیجه حافظه ، $O(1)$ است. ▷

سوال های ترکیبی

مسئله‌ی ۱۲. غیرممکن

داده ساختاری طراحی کنید که بتواند اعمال push، pop، و find min را در $O(1)$ انجام دهد. آیا می توان داده ساختار را به گونه ای تغییر داد که عمل delete min را نیز در $O(1)$ انجام دهد؟

حل.
دو پشته a و b را در نظر بگیرید که اعضای a همان اعضای ورودی هستند و

$$b[i] = min_{j \leqslant i} a[j]$$

برای اثبات این که نمی توان عمل delte min را نیز اضافه کرد، فرض کنید می توان اضافه کرد و سعی کنید در $O(n)$ اعداد را مرتب کنید.
▷

مسئله‌ی ۱۳. صف سریع با پشته

با استفاده از ۲ پشته، داده ساختار صف را طوری پیاده سازی کنید که هزینه سرشکن اعمال درج و حذف، $O(1)$ باشد.

حل.
برای درج یک عدد آن را در پشته اول درج می کنیم.
ترتیب حذف عناصر در داده ساختار صف، برعکس داده ساختار پشته است. برای این که بتوان این ترتیب را شبیه سازی کرد، می توان عناصر را از پشته اول حذف و در پشته دوم درج کرد.
به طور دقیق تر هنگام حذف، اگر پشته دوم خالی بود، تمامی عناصر موجود در پشته اول را وارد پشته دوم می کنیم و سپش عنصر رویی را حذف می کنیم. اگر هم که پشته دوم خالی نبود، عنصر

رویی آن را حذف می کنیم.
برای اثبات این که زمان سرشکن $O(1)$ است، دقت کنید که هر عنصر حداکثر یک بار از پشته اول به دوم می رود، در نتیجه چون هزینه خالی کردن پشته اول، هنگامی که i عنصر دارد، $O(i)$ است (یعنی برای هر عنصر، به طور متوسط $O(1)$)، کافیست که هنگام درج هر عنصر، "سکه" مورد نظر را روی آن قرار دهیم تا بتواند هزینه جابجایی از پشته اول به دوم را بپردازد. ◁

موفق باشید :)