پاسخ تمرین سری سوم درس ساختمان داده‌ها

1. از عناصر پایین‌تر شروع می‌کنیم و تا عناصر بالاتر پیش می‌رویم. هر عنصر را به سمت پایین heapify می‌کنیم. (n/2) عنصر پایین‌ترین سطح (سطح ۰) تغییری نمی‌کنند. (n/4) عنصر سطح ۱ هرکدام حداکثر به اندازه ۱ جابه‌جایی دارند. (n/8) عنصر سطح ۲ هرکدام حداکثر به اندازه ۲ جابه‌جایی دارند، و به همین ترتیب. بعد از اتمام انجام جابه‌جایی‌ها در یک سطح، می‌دانیم رئوسی که در سطوح پایین‌تر هستند، نیازی به جابه‌جایی مجدد ندارند.

بنابراین برای تعداد جابه‌جایی‌ها داریم:

$$\frac{n}{4} * 1 + \frac{n}{8} * 2 + \cdots + \frac{n}{n} * logn$$

این رابطه را می‌توان به شکل زیر نوشت:

$$\sum_{h=1}^{logn} \frac{n}{2^{h+1}} * O(h) = O\left(n * \sum_{h=1}^{logn} \frac{h}{2^{h+1}}\right) \leq O(n * \sum_{h=1}^{\infty} \frac{h}{2^h})$$

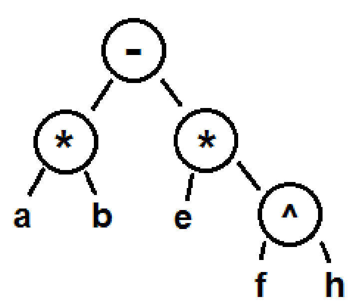
عبارت $\sum_{h=1}^{\infty} \frac{h}{2^h}$ را می‌توان به صورت زیر نوشت:

$$\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right) + \left(\frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots\right) + \cdots = \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 2$$

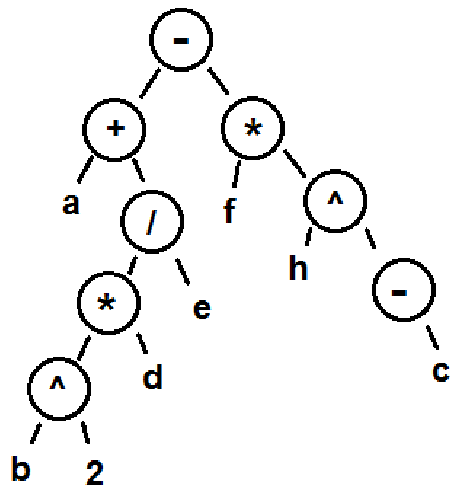
بنابراین کل جابه‌جایی‌ها هزینه‌ای برابر با $O(2 * n) = O(n)$ خواهد داشت.

2. درخت عبارت در هر دو حالت رسم شده است.

الف



ب



3-درخت بزرگ راA و درخت کوچک B می‌نامیم. فرض می‌کنیم دو درخت max-heap هستند. درخت Aرا حاوی k راس و درخت B را حاوی N راس فرض می‌کنیم.

از گوشه پایین چپ A به اندازه و شکل درخت B جدا می‌کنیم و آن را C می‌نامیم.

درخت‌های B و C را با هزینه زمانی $O(\log n)$ مخلوط می‌کنیم. مخلوط کردن دو درخت کامل و هم اندازه به این صورت است که درخت‌ها را کنار هم قرار داده و درست مانند وقتی که از یک درخت 2*n+1 تایی extract max کرده ایم، پایین‌ترین رأس سمت راست از درخت راست را در ریشه قرار داده و ریشه دو درخت B و C را فرزندان این ریشه جدید قرار می‌دهیم. حال این ریشه جدید را به سمت پایین heapify می‌کنیم.

درخت به‌دست آمده با ارتفاع یکی بیشتر از Bو Cاست. این درخت را D می‌نامیم. D را در گوشه پایین چپ درخت Aقرار می‌دهیم. در جایی که قبلا درخت C قرار داشت. (در این حالت قسمتی از درخت ارتفاعی بیشتر از بقیه درخت دارد به طور دقیق‌تر به اندازه تعداد رئوس Bدر ارتفاع جدید ظاهر می‌شود.) حال راس ریشه درخت D راE می‌نامیم. E را heapify می‌کنیم به سمت ریشه درخت بزرگ. با هر جابه‌جایی(جابه‌جایی E با پدرش) پدر آن را تا پایین درخت heapify به سمت پایین می‌کنیم. توجه داریم که این کار لازم است زیرا در درخت D که در سطوح پایینی درخت A قرار گرفته، رئوسی وجود دارد که قبلا در درخت B بودند، بنابراین ممکن است پدر راس E همچنان از رئوسی که داخل درخت D هستند، کوچکتر باشد و نیاز باشد که به پایین درخت برود. این کار را ادامه می‌دهیم تا راس E را تا ریشه heapify کنیم. درخت حاصل حاوی همه رئوس است و خصوصیات درخت heap را داراست.

هزینه زمانی کل= هزینه زمانی ساخت درخت D + هزینه heapify کردن راس E و پدرانش=

$$O(logn) + O(logk) + (O(\log n) + O(\log(n+1) + \cdots + O(\log(k)))$$

$$O(logn) + O(logk) + (O(\log(\mathrm{n}+1)) + O(logk)) * (O(logk) - O(\log(n+1)+1))/2$$

$$O(log^2 k)$$

-4

الف) به زیردرخت سمت راست ریشه (رئوس بزرگتر از ریشه) می‌رویم و یال‌های به سمت چپ را طی می‌کنیم (به سمت کوچکترین اعداد دراین زیر درخت پیش می‌رویم). این کار را تا جایی ادامه می‌دهیم که رأسی باشد که یال چپ (عدد کوچکتر از خود) نداشته باشد. این رأس حاوی کوچکترین عدد بزرگتر از ریشه است. بزرگتر از ریشه است زیرا در زیردرخت راست قرار

دارد و کوچکترین است زیرا در این زیر درخت همواره یال‌های به سمت چپ را طی کردیم. همچنین خود رأس مورد نظر فرزند چپ ندارد.

ب) زیردرخت سمت چپ راسA را LA و زیردرخت سمت راستش را RA می‌نامیم. از ریشه شروع می‌کنیم. ارزش رئوس Lroot کمتر از root و ارزش root کمتر از Rroot است. اگر k<num of nodes in Lroot+1 آنگاه زیر درخت Rroot را دور ریخته و جستجوی kامین کوچکترین عنصر را در Lrootادامه می‌دهیم.

اگر k=num of nodes in Lroot +1 جواب ریشه درخت(root) است.

اگر k > num of nodes in Lroot + 1 بود، زیردرخت چپ را دور ریخته و جستجوی ((k – (num of nodes in Lroot + 1) امین کوچکترین عنصر را درRroot ادامه می‌دهیم.

این کار را ادامه می‌دهیم تا به رأس مورد نظر برسیم

پیچیدگی زمانی در بدترین حالت $O(num\ of\ nodes\ in\ tree)$ است .

در حالتی که درخت متوازن باشد $O(\log(num\ of\ nodes\ in\ tree))$است زیرا هربار نصف راس های درخت دور ریخته می‌شوند.

5- اولین حرف نشان دهنده ریشه است. کلیدهایی که ارزش کمتر یا مساوی از ریشه دارند زیر درخت چپ و کلیدهایی که ارزش بیشتری دارند زیردرخت راست را تشکیل می‌دهند. ساخت هرکدام از زیردرخت‌ها به طور بازگشتی از هر زیررشته به دست می‌آید.

پیچیدگی زمانی:

پیدا کردن نقطه جداشدن زیردرخت راست و چپ یک ریشه از یک رشته را می‌توان به صورت یک جستجوی دودویی در ادامه رشته انجام داد. با این که رئوس به طور مرتب نیستند، از ابتدای رشته تا مرز جدایی رئوس دو زیردرخت، همه رئوس کوچکتر از ریشه هستند. به بیان واضح‌تر ابتدا نقطه میانی رشته را میبینیم و مقدارش را با ریشه(اولین حرف در رشته) مقایسه می‌کنیم. اگر این مقدار ارزشی کمتر از ریشه داشت، می‌دانیم که همه حروف سمت چپ این حرف نیز مقداری کمتر از ریشه دارند، پس برای یافتن مرز جدایی دو زیر درخت، در نیمه سمت راست رشته جستجو می‌کنیم. اگر حرف میانی ارزشی بیشتر از ریشه داشت هم با استدلال مشابه، باید مرز جدایی را در نیمه سمت چپ رشته جستجو کرد.

هزینه =$O(\log(length\ of\ remaining\ string))$

$$T(n) = logn + T(l) + T(n - l)$$

درحالت متوسط:

$$T(n) = logn + 2 * T\left(\frac{n}{2}\right) = n$$

این راه حل در زمانی بهینه است که درخت متوازن باشد. در غیر این صورت راه بهینه نیست. از آنجایی که درخت متوازن مد نظر بوده، این راه مناسب است.

-6

الف) فشرده‌سازی با اتلاف در مقایسه با فشرده‌سازی بی‌اتلاف فایلی با حجم کمتر تحویل می‌دهد، ولی ممکن است تعدادی از داده‌ها از دست برود. فشرده‌سازی بی اتلاف همه داده‌ها را نگه می‌دارد. و داده ای حذف نخواهد شد،ولی حجم زیادی نسبت به فشرده‌سازی

با اتلاف دارد.از فشرده‌سازی با اتلاف در تصاویر و صداها استفاده می‌شود. با حذف صداهای غیرقابل شنیدن و یا پایین آوردن کیفیت عکس، محتوای کلی را تغییر می‌دهد ولی تغییرات برای ما محسوس نخواهد بود. با این کارفایلی با حجم کمتر به ما ارائه می‌دهد.

ب) فایل در دو دور(2 passes) باید خوانده‌شود. درخت را هم باید همراه فایل فشرده شده منتقل کنیم که حجمی مضاعف اشغال خواهد کرد.

7-

A	B	C	D	E	F	G	H
1*400	5*10	4*30	5*11	2*200	5*12	5*11	4*40
0	11111	1101	11110	10	11100	11101	1100

4*40+5*11+5*12+2*200+5*11+4*30+5*10+1*400 = 1300