به نام خدا

ساختمان داده‌ها و الگوریتم‌ها (۴۰۲۵۴)

دانشگاه صنعتی شریف — تمرین چهارم

مدرس: دکتر مهدی صفرنژاد — انتشار: ۱۴ آذر ۱۴۰۰

درهم‌سازی و مرتب‌سازی سریع

سؤالات را با دقت بخوانید و روی همه آن‌ها وقت بگذارید. تمرین‌های تئوری تحویل گرفته نمی‌شوند اما از آن‌ها سؤالات کوییز مشخص می‌شود. بنابراین روی سؤالات به خوبی فکر کنید و در کلاس‌های حل تمرین مربوطه شرکت کنید.

سؤال ۱. شیوه خاصی از تابع درهم‌سازی به روش تقسیم را در نظر بگیرید که در آن $h(k) = k \bmod m$ که در آن $m = 2^p - 1$ و k هم یک رشته از کاراکترهاست که بیانگر عددی در مبنای 2^p است. نشان دهید که اگر دو رشته x و y داشته باشیم که با جابه‌جا کردن کاراکترهای یکی از آن‌ها بتوانیم به دیگری برسیم (یکی از آن‌ها جایگشتی از دیگری باشد)، هر دوی این رشته‌ها با تابع درهم‌سازی ذکر شده به یک خانه نگاشت می‌شوند. مثالی از یک کاربرد واقعی بیاورید که وجود چنین خاصیتی نامطلوب باشد.

پاسخ: نشان می‌دهیم که هش هر رشته در روش گفته شده برابر جمع ارقامش به پیمانه $2^p - 1$ خواهد بود. برای این کار از استقرا استفاده می‌کنیم. برای پایه استقرا یک رشته تک کاراکتری را در نظر بگیرید. در این صورت مقدار آن به پیمانه $m = 2^p - 1$ محاسبه می‌شود. پس برای پایه استقرا حکم برقرار است. حال برای گام استقرا فرض کنید رشته $w = w_1w_2$ باشد که از n کاراکتر تشکیل شده است و w_1 شامل $n-1$ کاراکتر پرارزش تر باشد و w_2 هم کاراکتر آخر آن باشد. در این صورت

$$h(w) = h(w_1)2^p + h(w_2) \bmod 2^p - 1$$

$$h(w) = h(w_1) + h(w_2)$$

این یعنی که برای یک رشته با طول دلخواه، حاصل تابع درهم ساز معادل حاصل جمع هش تمامی آن به جز رقم آخر و هش رقم آخر به صورت جداگانه است. این موضوع نشان می‌دهد که هش هر رقم به طور جداگانه تاثیر دارد و در نتیجه اگر یک عدد k رقمی به فرم $x = x_1x_2\cdots x_k$ داشته باشیم

$$h(x) = h(x_k) + h(x_1 \cdots x_{k-1}) = h(x_k) + h(x_{k-1}) + h(x_1 \cdots x_{k-2}) = \cdots = h(x_1) + \cdots + h(x_k)$$

خواهد بود.

برای مثال حالتی که چنین سیستمی مشکل ساز می‌شود، فرض کنید که یک دیکشنری انگلیسی طراحی می‌کنید که لغات براساس این تابع در آن ذخیره می‌شوند و کاراکترهای مورد استفاده ما هم کدهای ASCII باشند و در نتیجه مبنا $2^8 = 256$ و $m = 255$ باشد. در این صورت مثلا کلماتی نظیر STOP,TOPS,SPOT,POTS که همگی معنی دار هم هستند به یک slot نگاشت خواهند شد.

سؤال ۲. تابع درهم‌سازی یکنواخت ساده h را در نظر بگیرید. فرض کنید قرار است n کلید یکتا را با استفاده از این تابع به یک جدول با اندازه m، map کنیم. تعداد تصادم‌ها به طور متوسط چقدر خواهد بود؟

پاسخ: فرض کنید کلیدها به صورت $k_1, k_2, ..., k_n$ مرتب شده باشند. حالا X_i را برابر با تعداد k_j هایی قرار می‌دهیم که $h(k_j) = h(k_i)$ و $k_j > k_i$. بخاطر یکنواخت بودن درهم‌سازی خواهیم داشت:

$$X_i = \sum_{j>i} P(h(k_j) = h(k_i)) = \sum_{j>i} \frac{1}{m} = \frac{(n-i)}{m}$$

از طرفی می‌توان گفت مجموع تعداد تصادم‌ها برابر است با مجموع تمام X_i ها. بنابراین برای تعداد تصادم‌ها خواهیم داشت:

$$\sum_{i=1}^{n} X_i = \sum_{i=1}^{n} \frac{n-i}{m} = \frac{n^2 - \frac{n(n+1)}{2}}{m} = \frac{n^2-n}{2m}$$

سؤال ۳. هش دوگانه به این صورت است که دو تابع h_1 و h_2 را به عنوان توابع هش در نظر می‌گیریم. حالا برای هش کردن یک کلید k به این صورت عمل می‌کنیم که اگر خانه‌ی $h_1(k)$ قبلا پر شده بود سراغ خانه‌ی $h_1(k) + h_2(k)$ می‌رویم. در صورت پر بودن این خانه سراغ خانه $h_1(k) + 2h_2(k)$ می‌رویم و ...
فرض کنید یک تابع هش دوگانه به صورت $h(k,i) = (h_1(k) + ih_2(k))\%m$ داریم. حالا اگر اندازه جدول هش m باشد نشان دهید که اگر ب.م.م $h_2(k)$ و m برابر با $d \geq 1$ باشد، یک جستجوی ناموفق برای کلید k، $\frac{1}{d}$ خانه‌های جدول را قبل از برگشتن به خانه‌ی $h_1(k)$ بررسی می‌کند.

پاسخ: قرار می‌دهیم $h_2(k) = dc_1, m = dc_2$ گفتیم که سایز جدول برابر با m است پس $\frac{1}{d}$ عناصر جدول برابر با $\frac{m}{d} = c_2$ خواهد بود. بنابراین خانه‌ای که ما در این مرحله بررسی می‌کنیم برابر است با:

$$[h_1(k) + (\frac{m}{d} h_2(k))]\%m = [h_1(k) + \frac{m}{d}(dc_1)]\%m = h_1(k)$$

سؤال ۴. در جدول درهم سازی با استفاده از روش وارسی خطی، تابع درهم سازی برای جدولی با اندازه هشت به صورت زیر است:

key	A	B	C	D	E	F	G	H
hash	۲	۶	۲	۴	۴	۵	۴	۱

اگر جدول درهم سازی در ابتدا تهی باشد، به چند حالت میتوان این عناصر را در جدول درج کرد تا در نهایت جدول زیر تولید شود؟

i	۰	۱	۲	۳	۴	۵	۶	۷
T(i)	G	H	A	C	D	E	B	F

پاسخ:

$Hash(A) = 2, Hash(C) = 2$
$T(2) = A, T(3) = C$

بنابراین به دو مورد زیر C بعد از A هش میشود.

$Hash(D) = 4, Hash(E) = 4$
$T(4) = D, T(5) = E$

و E بعد از D هش میشود.

$Hash(E) = 4, Hash(B) = 6, Hash(F) = 5$
$T(5) = E, T(6) = B, T(7) = F$

و F بعد از E و B هش میشود.

$Hash(D) = 4, Hash(E) = 4, Hash(G) = 4$
$Hash(B) = 6, Hash(F) = 5$
$T(5) = E, T(6) = B$
$T(7) = F, T(0) = G$

و G باید بعد از F و E و B هش شود.

حالا تعداد جایگشت های مختلف را محاسبه میکنیم. H هیچ محدودیتی نداشته است پس ۸ جایگشت مختلف دارد. A و C هم با توجه به یکی از محدودیت ها دارای $C(7,2)$ یعنی ۲۱ جایگشت مختلف هستند. با توجه به شروط دو و سه و چهار هم متوجه میشویم که ترتیب هش شدن به صورت D و E و F و G است و B هم قبل از F باید هش شود پس ۳ حالت دارد.

برابراین به طور کلی $8 * 3 * 21 = 504$ حالت مختلف برای ساختن این آرایه داریم.

سؤال ۵. الگوریتمی ارائه دهید که آرایه‌ای از اعداد یکتا ورودی بگیرد و در خروجی تعداد اعداد بزرگتر سمت راست هر اندیس را خروجی دهد. مثال:

$input = [4, 6, 3, 9, 7, 10]$
$output = [4, 3, 3, 1, 1, 0]$

پاسخ: یک راه حل ساده این است که برای هر عنصر آرایه، همه عناصر را بزرگتر از آن در سمت راستش بشمارید. این الگوریتم در زمان $O(n^2)$ اجرا می شود.

راه حل بهینه این مساله شبیه پیداکردن تعداد نابه‌جایی های یک آرایه با استفاده از مرتب‌سازی ادغامی است. با استفاده از این الگوریتم می‌توانیم پیچیدگی زمانی را به $O(nlog(n))$ کاهش دهیم. آرایه را به ترتیب نزولی مرتب کنید و یک نقشه برای ذخیره تعداد اعداد بزرگتر از هر عنصر مجزا نگه دارید. در هر مرحله از مرتب سازی ادغامی جدول هش را بروزرسانی می‌کنیم، یعنی هر گاه به دو تایی‌ای رسیدیم که $i > j$ و $array[i] > array[j]$ (برعکس یه نابه‌جایی)یک واحد به اندیس $array[j]$ ام جدول هش اضافه می‌کنیم. با توجه به استفاده از $hashmap$ ، فضای مورد استفاده الگوریتم هم از $O(n)$ خواهد بود. (توجه داشته باشید که با توجه به اینکه در $hashmap$ کلید هایمان اعداد موجود در آرایه هستند و برای آن‌ها محدودیتی نداریم نمی‌توانیم مانند حالت عادی از یک آرایه $counter$ استفاده کنیم.)

سؤال ۶. الگوریتمی از $O(nlog(n))$ بیابید که در رشته s به طول n تعداد زیررشته‌های آینه‌ای را پیدا کند. زیررشته‌ی آینه‌ای رشته‌ای است که از هر دو طرف به یک شکل خوانده می‌شود.

پاسخ: اول از همه بین همه‌ی کاراکترها و همین‌طور ابتدا و انتهای رشته یک کاراکتر دلخواه درج می‌کنیم تا طول رشته حتما فرد بشود. علت این کار ساده شدن فرایند جستجوی دودویی در ادامه مسئله است. حالا به هر کاراکتر یک عدد یکتا نسبت می‌دهیم و تابع هش را برای رشته‌ی $s = s_0s_1...s_{n-1}s_n$ به شکل زیر تعریف می‌کنیم:

$$h(s) = p^{n-1}s_0 + p^{n-2}s_1 + ... + ps_{n-1} + s_n$$

آرایه T را به طبق رابطه بازگشتی $T[0] = s_0, T[i] = s_i + pT[i-1]$ پر می‌کنیم. با این کار برای مقدار تابع هش هر زیرآرایه‌ی $s_{i,j}$ خواهیم داشت: $h(s_{i,j}) = T[j] - T[i-1]p^{(j-i)+1}$ همین کار را برای معکوس رشته s و آرایه Q انجام می‌دهیم. سپس سراغ گام نهایی می‌رویم. می‌دانیم که هر زیررشته‌ی آینه‌ای یک کاراکتر میانی خواهد داشت. بنابراین روی تمام رشته لوپ می‌زنیم و هر بار یک کاراکتر را، کاراکتر میانی در نظر می‌گیریم و تمام زیررشته‌های آینه‌ای که کاراکتر انتخاب شده کاراکتر میانی آن است را پیدا می‌کنیم. برای این کار از آرایه‌هایی که قبلا حساب کردیم و همچنین جستجوی باینری استفاده می‌کنیم. روش کلی به این صورت است که اگر s و t دو زیرشته با کاراکتر میانی مشترک باشند و همچنین t زیررشته s باشد، تنها در صورتی s آینه‌ای است که t هم آینه‌ای باشد. همچنین تمام زیر رشته‌های t هم آینه‌ای خواهند بود. بنابراین با در نظر گرفتن یک کاراکتر میانی و استفاده از جستجوی دودویی، می‌توان ابتدا زیررشته‌ای را بررسی کرد که ابتدای آن، وسط بازه‌ی ابتدای رشته تا کاراکتر میانی‌در نظر گرفته شده است و مکان انتهای آن نیز متناظر با ابتدای آن در سمت دیگر کاراکتر میانی است. حالا اگر هش این زیررشته با هش وارون آن برابر باشد متقارن است و می‌توانیم سراغ رشته‌های بزرگ‌تر که این رشته زیررشته آن‌ها است برویم و در غیر این صورت جستجوی دودویی را به زیررشته‌های کوچک‌تر محدود کنیم. همچنین دیدیم که با استفاده از آرایه‌هایی که قبلا حساب کردیم، می‌توانیم هش

یک زیررشته یا وارون آن را در $O(1)$ حساب کنیم.
برای پیچیدگی زمانی هم می‌دانیم که جستجوی دودویی در زمان $O(log(n))$ انجام می‌شود و چون n بار از جستجویی دودویی استفاده می‌کنیم، پس پیچیدگی زمانی برابر با $O(nlog(n))$ خواهد بود.
مثلا رشته $\#a\#f\#c\#a\#c\#f\#b\#$ و دومین کاراکتر a از سمت چپ را به عنوان کاراکتر میانی در نظر بگیرید. برای جستجوی باینری به این صورت عمل می‌کنیم که در ابتدا وسط زیررشته $\#a\#f\#c\#$ یعنی f را به عنوان مبدا باینری‌سرچ در نظر می‌گیریم و آینه‌ای بودن زیر رشته $f\#c\#a\#c\#f$ را بررسی می‌کنیم. چون این زیررشته آینه‌ای است، پس تمام زیررشته‌های آن نیز آینه‌ای هستند و جستجو را برای زیردنباله‌های بلندتر که کاراکتر میانی‌آنها a مذکور است انجام می‌دهیم که در اینجا زیرآرایه‌ی $a\#f\#c\#a\#c\#f\#b$ خواهد بود. طبق چیزی که گفته شد، هر بار یکی از کاراکترهای آرایه به عنوان کاراکتر میانی در نظر گرفته خواهد شد و جستجوی دودویی با روشی که توضیح دادیم انجام می‌شود.

سؤال ۷. آرایه‌ای از اعداد داریم که در آن اعداد تکراری زیادی وجود دارد. الگوریتمی ارائه دهید که چنین آرایه‌ای را در اردر زمانی بهتر از $O(nlog(n))$ مرتب‌سازی نماید.

پاسخ: با استفاده از جدول هش این مسئله را حل می‌کنیم جدولی طراحی می‌کنیم که دارای دو ستون می‌باشد. در ستون اول اعداد یکتای آرایه نوشته شده و در ستون مقابل آن، فرکانس آن را ذخیره می‌کنیم. . الگوریتم به روش زیر خواهد بود.

- یک بار آرایه را پیمایش می‌کنیم و فرکانس هر عدد را در جدول هش ذخیره می‌کنیم.
- اعداد یکتا در جدول هش به روش دلخواه مرتب می‌کنیم.
- آرایه را بر اساس ترتیب به دست آمده و فرکانس هر عدد، که در جدول هش ذخیره شده است، بازنویسی می‌کنیم.

با این فرض که در آرایه در مجموع k عدد یکتا وجود دارد، الگوریتم بالا دارای پیچیدگی زمانی $O(n.log(k))$ و نیازمند فضای ذخیره سازی از $O(k)$ می‌باشد.

سؤال ۸. الگوریتمی ارائه دهید که پس از ورودی گرفتن یک آرایه، وجود یا عدم وجود زیرآرایه‌ای از آن که مجموع عناصر آن صفر باشد را در زمان خطی تشخیص دهد.

پاسخ:

پاسخ: این مسئله به راحتی با استفاده از هش کردن حل خواهد شد. نخست مجموعه‌ی S را تعریف می‌کنیم حال از ابتدای آرایه شروع به پیمایش می‌کنیم و مجموع اعداد ویزیت شده تا آن لحظه را به مجموعه‌ی S اضافه می‌کنیم و در هر مرحله اگر مجموع به دست آمده، قبلا به S اضافه شده بود (به عنوان مثال x) ، $True$ را $return$ و مسئله را تمام می‌کنیم، زیرا از آنجایی که مجموع عناصر یک بار به x رسید، سپس تغییراتی کرد و دوباره برابر x شد، قطعا زیرآرایه‌ای که انتهای آن برابر index فعلی می‌باشد وجود دارد که مجموع عناصر آن برابر صفر است.
از آنجایی که در این الگوریتم صرفا یک بار آرایه را پیمایش کردیم، پیچیدگی زمانی خطی داشته و از $O(n)$ می‌باشد.
مثال:

$$array = [1, 2, 3, -2, 5, -4, -2, 7]$$

حال مجموعه‌ی S را تعریف کرده و هر بار مجموع اعداد صفر تا i را در حلقه‌ی 0 تا n ، به آن اضافه می‌کنیم.

$i = 0 s = 1$

$i = 1 s = 1, 3$

$i = 2 s = 1, 3, 6$

$i = 3 s = 1, 3, 6, 4$

$i = 4 s = 1, 3, 6, 4, 9$

$i = 5 s = 1, 3, 6, 4, 9, 5$

$i = 6 s = 1, 3, 6, 4, 9, 5, 3$

در اینجا می‌بینیم که در دنباله $partialsum$ عدد ۳ دوبار تکرار می‌شود. در واقع می‌توان دید که مجموع اعداد بین اولین ۳ تا دومین ۳ برابر صفر خواهند بود. $3 + (-2) + 5 + (-4) + (-2) = 0$

موفق باشید