موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

ساختمان داده و طراحی الگوریتم

(حل تشریحی سوالات دولتی ۱۳۹۷)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتاب مرجع

توماس اچ کورمن، چارلز ای لیزرسان، رونالد ال ریوست و کلیفورد استین

ابوالفضل گیلک

کد کنترل

333

E

333E

«اگر دانشگاه اصلاح شود مملکت اصلاح می‌شود.»
امام خمینی (ره)

صبح جمعه
۹۷/۲/۷

جمهوری اسلامی ایران
وزارت علوم، تحقیقات و فنّاوری
سازمان سنجش آموزش کشور

آزمون ورودی دوره‌های کارشناسی‌ارشد ناپیوستهٔ داخل ـ سال ۱۳۹۷

مهندسی کامپیوتر ـ کد (۱۲۷۷)

تعداد سؤال: ۱۴۰

مدت پاسخگویی: ۲۵۵ دقیقه

عنوان مواد امتحانی، تعداد و شمارهٔ سؤالات

ردیف	مواد امتحانی	تعداد سؤال	از شمارهٔ	تا شمارهٔ
۱	زبان عمومی و تخصصی (انگلیسی)	۳۰	۱	۳۰
۲	ریاضیات (ریاضی عمومی (۱و۲)، معادلات دیفرانسیل، آمار و احتمال مهندسی، ریاضیات گسسته)	۲۰	۳۱	۵۰
۳	دروس تخصصی مشترک (ساختمان داده‌ها و طراحی الگوریتم‌ها، نظریه زبان‌ها و ماشین‌ها، مدارهای منطقی، معماری کامپیوتر، سیستم عامل و شبکه‌های کامپیوتری)	۳۰	۵۱	۸۰
۴	دروس تخصصی معماری سیستم‌های کامپیوتری (مدارهای الکتریکی، الکترونیک دیجیتال و VLSI، سیگنال‌ها و سیستم‌ها)	۲۰	۸۱	۱۰۰
۵	دروس تخصصی نرم‌افزار، شبکه‌های کامپیوتری، رایانش امن (کامپایلر، پایگاه داده‌ها، هوش مصنوعی)	۲۰	۱۰۱	۱۲۰
۶	دروس تخصصی هوش مصنوعی و رباتیکز (مدارهای الکتریکی، هوش مصنوعی، سیگنال‌ها و سیستم‌ها)	۲۰	۱۲۱	۱۴۰

این آزمون نمرهٔ منفی دارد.

استفاده از ماشین‌حساب مجاز نیست.

دروس تخصصی مشترک (ساختمان داده‌ها و طراحی الگوریتم‌ها، نظریه زبان‌ها و ماشین‌ها، مدارهای منطقی، معماری کامپیوتر، سیستم عامل و شبکه‌های کامپیوتری):

۵۱- یک درخت دودویی جست‌وجو شامل n عدد و ارتفاع $O(\log n)$ در اختیار داریم. به ازای هر گره در درخت فوق تعداد نوادگان آن گره به عنوان اطلاعات اضافه، ذخیره شده است. کدام مورد را در زمان $O(\log n)$ نمی‌توان پاسخ داد؟

۱) تعداد اعداد کوچک‌تر از عدد داده شدهٔ a

۲) تعداد اعداد ذخیره شده در درخت که در بازهٔ داده شدهٔ $[a.b]$ قرار دارند.

۳) میانهٔ اعداد ذخیره شده در درخت که در بازهٔ داده شدهٔ $[a.b]$ قرار دارند.

۴) میانگین اعداد ذخیره شده در درخت که در بازهٔ داده شدهٔ $[a.b]$ قرار دارند.

۵۲- آرایهٔ یک بعدی A، شامل n عدد صفر و یک است. اگر به ازای هر صفر، اولین یک سمت چپ (با اندیس کمتر) و به ازای هر یک، اولین صفر سمت چپ آن را پیدا کنیم، هزینهٔ سرشکن این محاسبه برای هر عدد، کدام است؟ (بهترین پاسخ را انتخاب کنید.)

۱) $O(1)$

۲) $O(n)$

۳) $O(\log n)$

۴) $O(\log \log n)$

۵۳- جواب رابطهٔ بازگشتی $T(n) = T(\sqrt{n}) + O(\log\log n)$. کدام است؟

۱) $O(\log n)$
۲) $O(\log^2 n)$
۳) $O(\log\log n)$
۴) $O(\log^2 \log n)$

۵۴- آرایهٔ A از n عدد دلخواه متمایز تشکیل شده و k یک عدد از پیش مشخص است. فرض کنید عملیات sort(i) به ازای $1 \le i \le n-k+1$، زیرآرایهٔ $A[i..i+k-1]$ را مرتب می‌کند. در بدترین حالت چند عملیات sort برای مرتب کردن آرایهٔ A لازم است؟ (بهترین پاسخ را انتخاب کنید.)

۱) $O(n^2/k)$
۲) $O(n^2/k^2)$
۳) $O(n\log n)$
۴) $O(n\log_k n)$

۵۵- یک جدول درهم‌ساز داریم. فرض کنید برای رفع مشکل تصادم از روش وارسی خطی استفاده شده است. با درنظر گرفتن فرض یکنواختی تابع در هم‌ساز، کلید بعدی با چه احتمالی در خانهٔ دوم قرار می‌گیرد؟
(خانه‌های جدول از چپ به راست از ۱ تا ۱۸ شماره‌گذاری شده اند.)

۵		۷				۱۱		۲	۹		۱۴		۳		۱	۴	۶

۱) $\frac{۱}{۱۸}$
۲) $\frac{۵}{۱۸}$
۳) $\frac{۸}{۱۸}$
۴) $\frac{۱۰}{۱۸}$

۵۶- آرایهٔ A شامل n عدد داده شده است. همچنین یک جعبه سیاه داریم که به عنوان ورودی یک زیرمجموعهٔ $S \subseteq \{1,2,\ldots,n\}$ با اندازهٔ حداکثر k و یک عدد x را به عنوان ورودی می‌گیرد و اگر عدد $i \in S$ وجود داشت طوری که $A[i] = x$، مقدار یک را برمی‌گرداند و در غیر این صورت صفر برمی‌گرداند. با چند مرتبه استفاده از این جعبه سیاه می‌توانیم به ازای یک عدد دلخواه x در صورت وجود، اندیس y را که $A[y] = x$ است پیدا کنیم؟ (بهترین پاسخ را انتخاب کنید.)

۱) $O(n)$
۲) $O(n/k)$
۳) $O(\log n)$
۴) $O(n/k + \log k)$

۵۷- گراف وزن‌دار و همبند G را درنظر بگیرید (وزن‌ها مثبت هستند). وزن یک مسیر ساده (بدون رأس تکراری) در گراف را برابر وزن یالی که در مسیر کمترین وزن را دارد، تعریف می‌کنیم. در الگوریتم‌های بلمن ـ فورد و دایکسترا، $d[u]$ برابر سبک‌ترین مسیر سادهٔ به دست آمده تاکنون از مبدا درنظر گرفته می شود. اگر در این الگوریتم‌ها به ازای یال (u,v) با وزن $w(u,v)$ به‌روزرسانی را به این شکل تغییر دهیم که $d[v] = \min(w(u,v), d(u))$. کدام یک از دو الگوریتم فوق با تغییر انجام شده، همیشه درست کار می‌کند؟ (مقدار اولیهٔ $d(u)$ در هر دو الگوریتم برابر مثبت بی‌نهایت قرار داده می‌شود.)

۱) هر دو الگوریتم
۲) فقط الگوریتم دایکسترا
۳) فقط الگوریتم بلمن ـ فورد
۴) هیچ یک از این دو الگوریتم

۵۸- گراف وزن‌دار، همبند و بدون جهت $G=(V,E)$ را درنظر بگیرید. الگوریتم زیر را روی G اجرا می‌کنیم.

در ابتدا $M=(V,\{\})$ قرار می‌دهیم. سپس یال‌های G را به‌ترتیب دلخواه در M درج می‌کنیم. بعد از درج هر یال، اگر M دارای دور بود، به ازای هر دور در M سنگین‌ترین یال آن دور را حذف می‌کنیم. کدام گزاره‌ها درست هستند؟

a) M همیشه برابر درخت پوشای کمینه G است.

b) اگر یال‌ها به‌ترتیب وزن (از کوچک به بزرگ) درج شوند، M حتماً درخت پوشای کمینه خواهد بود.

۱) a درست ، b درست
۲) a نادرست ، b درست
۳) b نادرست ، a درست
۴) a نادرست ، b نادرست

۵۹- برای دنباله $X=\langle x_1,\ldots,x_n\rangle$ متشکل از اعداد متمایز، فرض کنید $LIS(X)$ بزرگ‌ترین زیردنبالهٔ صعودی X و $LIS(X,a)$ بزرگ‌ترین زیر دنبالهٔ صعودی X که عنصر آخر آن حداکثر a می‌باشد. چه تعداد از گزاره‌های زیر درست هستند؟

(در زیر $X_i=\langle x_1,\ldots,x_i\rangle$ و عملگر max دنبالهٔ با طول بزرگ‌تر را برمی‌گرداند.)

- $LIS(X_n)=\max_{i=1}^{n}(\langle LIS(X_{i-1},x_i),x_i\rangle)$
- $LIS(X_n)=\langle LIS(X_{n-1},x_n),x_n\rangle$
- $LIS(X_n)=\max(LIS(X_{n-1}),\langle LIS(X_{n-1},x_n),x_n\rangle)$

۱) ۰
۲) ۱
۳) ۲
۴) ۳

۶۰- فرض کنید برای ساخت درخت کد هافمن از الگوریتم زیر استفاده کنیم. حروف الفبا را به دو دستهٔ A و B به گونه‌ای افراز می‌کنیم که اختلاف تعداد تکرارهای حروف الفبا در A و B کمینه شود. به‌طور بازگشتی درخت کد هافمن را برای هر یک از این دو دسته می‌سازیم. سپس دو درخت به دست آمده برای A و B را به‌عنوان زیردرخت‌های ریشه قرار می‌دهیم. (اگر تعداد حروف الفبا ۱ باشد، درخت کد هافمن تک رأسی است.) اگر n تعداد حروف الفبا باشد، کوچک‌ترین مقدار n که برای آن الگوریتم فوق درخت بهینه را تولید نمی‌کند، کدام است؟

۱) ۲
۲) ۳
۳) ۴
۴) ۵

۵۱ گزینه (۴)

① پاسخ سریع:

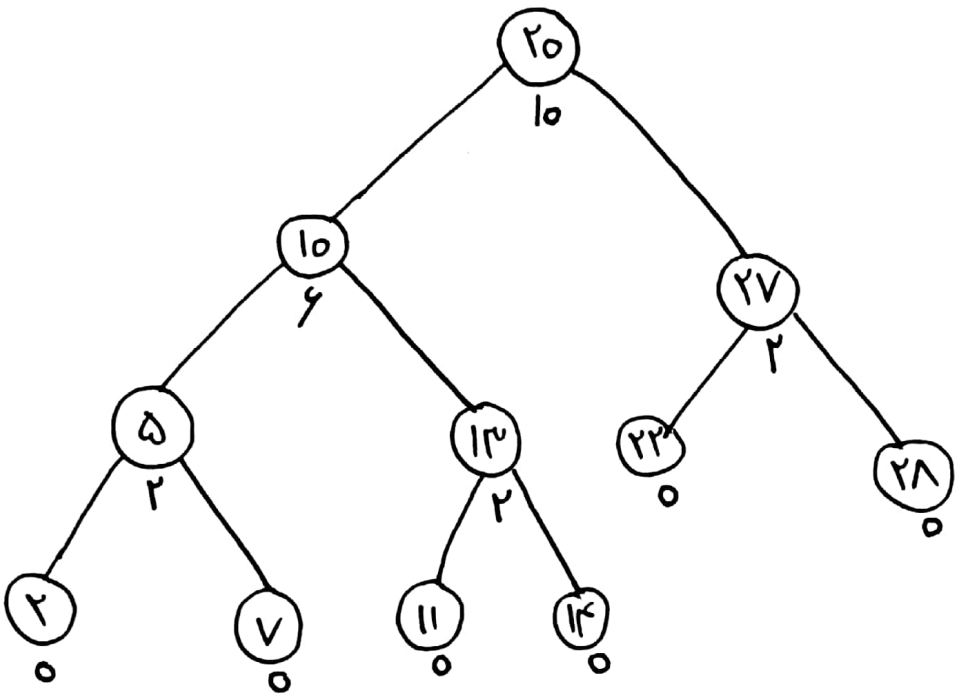
ابتدا توجه شما را به این مطلب جلب می کنم که تعداد نوادگان هر گره، به عنوان اطلاعات اضافی در کنار آن گره ذخیره شده است، بنابراین مسئله هایی که فقط به تعداد گره ها مربوط باشند، احتمالاً به سرعت قابل حل خواهند بود.

گزینه های (۱)، (۲) و (۳) فقط به تعداد اعداد بستگی دارند. مثلاً میانه، جایی است که تعداد گره های بزرگتر از آن با تعداد گره های کوچکتر از آن برابر (حتی المقدور برابر) باشد.

اما گزینه (۴) که میانگین یک سری از اعداد را می خواهد، فقط با داشتن تعداد آنها قابل حل نیست. پس می توان با اطمینان گفت که در بین این ۴ گزینه، این گزینه‌ی (۴) است که مرتبه‌ی زمانی بزرگتری دارد.

پس به عنوان یک مسأله‌ی چهارگزینه‌ای، به سرعت می توان به آن پاسخ داد. اما حالا حل تشریحی را هم مرور خواهیم کرد.

② پاسخ تشریحی:



از درخت جستجوی دودویی بالا به عنوان مثالی برای توضیح مساله در حالت کلی استفاده می کنیم.

همان طورکه می دانید، در یک درخت جستجوی دودویی، همه ی فرزندان سمت راست (نوادگانِ سمت راست) از ریشه بزرگتر هستند و همه ی نوادگان سمت چپ، از ریشه کوچکتر هستند.

ارتفاع درخت جستجوی دودویی، حداکثر $O(n)$ و حداقل $O(\log n)$ است و به طور متوسط هم ارتفاع $O(\log n)$ به دست می آید.

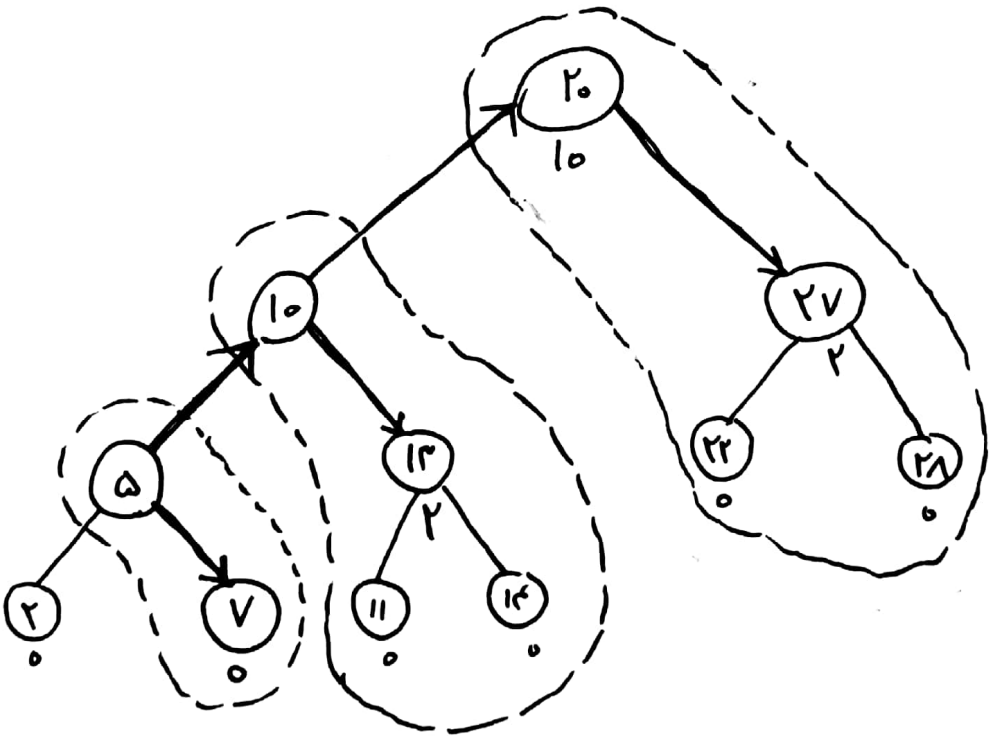
(مثلا اگر برای n گره، همه ی انواع درخت جستجوی دودویی را رسم کنید، معدلِ ارتفاعِ این درخت ها $O(\log n)$ است.)

طبق صورت سئوال ، درخت مورد بحث ما ارتفاع $O(\log n)$ دارد، و در ضمن تعداد نوادگان هر گره در کنارش ذخیره شده است.

حالا گره ذخیره شده‌ی $a=5$ را به عنوان مثال در نظر بگیرید. می‌خواهیم تعداد گره‌های بزرگتر یا مساوی a را بشماریم. این کار تقریباً شبیه یافتن گره مابعد است. (CLRS یا حل‌ی ۱۳)

ابتدا تعداد نوادگانِ فرزند راستِ a را (در صورت وجودِ این فرزند) در نظر می‌گیریم و ۲ واحد هم به آن اضافه می‌کنیم.

پس از درخت بالا می‌رویم و هرگاه به گره‌ای رسیدیم که فرزند چپ آن بودیم، تعداد نوادگانِ فرزند راستِ آن گره را هم به مجموع خودمان اضافه می‌کنیم البته ۲ واحد هم همیشه اضافه می‌کنیم.



تعداد اعداد بزرگتر مساوی ۵ $= (0+2)+(2+2)+(2+2) = 10$

دقت کنید که وقتی در گره x هستید، دسترسی به فرزند راست x از مرتبه‌ی $O(1)$ انجام می‌شود. در ضمن بالا رفتن از درخت، حداکثر چقدر ادامه خواهد یافت؟ به اندازه‌ی ارتفاع درخت - پس کل این محاسبات در زمان $O(h)$ انجام می‌شود که $h = O(\log n)$ است.

در یک مثال دیگر می‌خواهیم برای $a = 14$ تعداد گره‌های بزرگتر مساوی ۱۴ را حساب کنیم.

اولاً: تعداد نوادگانِ فرزند راستِ خودش را به علاوه‌ی ۲ کنید.

ثانیاً: شروع به بالا رفتن از درخت کنید. از گره (۱۴)

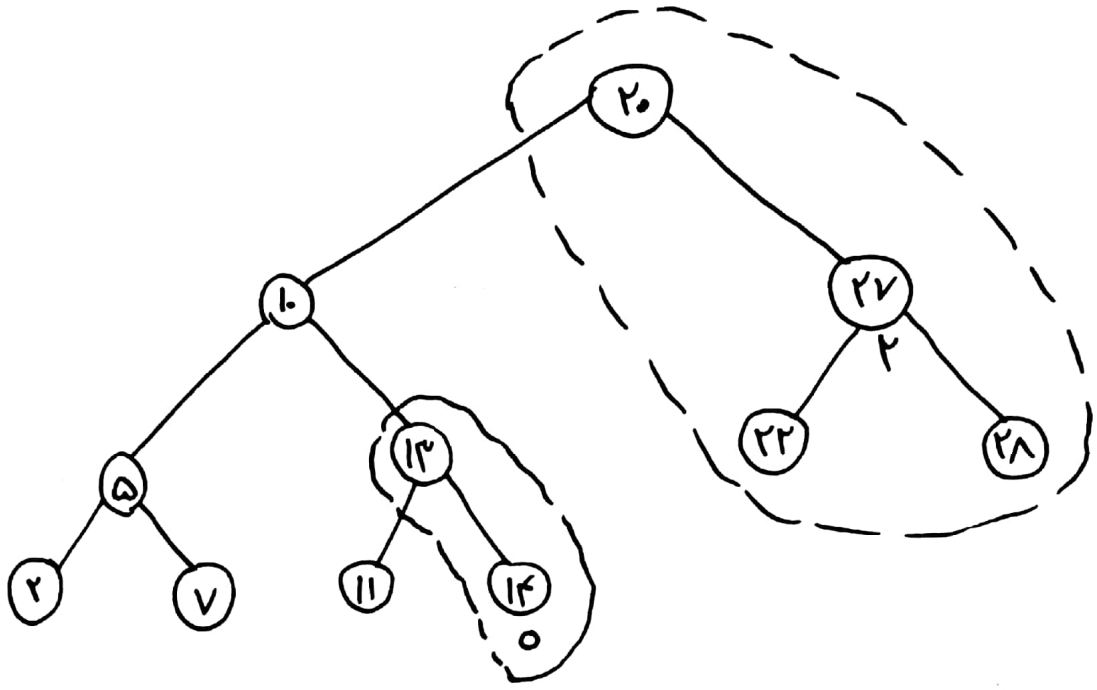
به گره (۱۵) می‌رسید. اما این گره مطلوب نیست چون شما فرزند راست آن هستید.

حالا از (۱۵) به (۲۰) می‌روید. این گره، مطلوب است چون شما فرزند چپ آن هستید. حالا:

تعداد نوادگانِ فرزند راستِ (۲۰) را به علاوه‌ی ۲ کنید.

دیگر بالاتر نمی‌توان رفت چون پدر (۲۰)، NIL است.

در مجموع داریم:



تعداد گره‌های بزرگتر مساوی ۱۴ $= (5+2) + (2+2) = 6$

نتیجه (الف) در مرتبه‌ی زمانی $O(\log n)$ می‌توان تعداد گره‌های بزرگتر یا مساوی گره‌ی ذخیره شده‌ی a را حساب کرد.

به روشی مشابه، می‌توان در مرتبه زمانی $O(\log n)$ تعداد گره‌های کوچکتر یا مساوی گره ذخیره شده‌ی b را حساب کرد.

درمورد گزینه (۲)

فرض کنید A مجموعه‌ی گره‌هایی باشد که بزرگتر مساوی a هستند.

$$A = \{x \in T \mid x \geq a\}$$

و B مجموعه‌ی گره‌هایی باشد که کوچکتر مساوی b هستند:

$$B = \{x \in T \mid x \leq b\}$$

واضح است که اگر $b < a$ باشد بازه‌ی $[a, b]$ تهی می‌شود پس صحبت منظور ما حالت $a < b$ است. هدف ما در گزینه‌ی (۲) یافتن تعداد اعضای $A \cap B$ است یعنی تعداد گره‌هایی که $a \leq x \leq b$ باشد.

به وضوح داریم: $A \cup B =$ همه گره‌ها

پس $|A \cup B| = n$ است.

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$\Rightarrow |A \cap B| = -n + |A| + |B|$$

در (الف) دیدیم که $|A|$ و $|B|$ در زمان $O(\log n)$ قابل محاسبه‌اند پس $|A \cap B|$ هم در زمان $2\log n + 1$ محاسبه می‌شود که همان $O(\log n)$ است.

با توجه به روند توضیح داده شده برای گزینه های (۱) و (۲)
یافتن میانه‌ی گره ها در بازه‌ی [a,b] به سادگی (البته نه خیلی واضح)
از مرتبه O(logn) انجام می‌شود. به عنوان مثال اگر a کوچکترین
و b بزرگترین گره باشد و [a,b] شامل تمام گره ها شود
باشد و از ریشه و توجه به تعداد گره های بزرگتر و کوچکتر آن
می‌توانید گره ای را بیابید که تعداد گره های بزرگتر و کوچکتر آن
تا حد امکان برابر باشد.

مرتبه زمانی گزینه (۴)

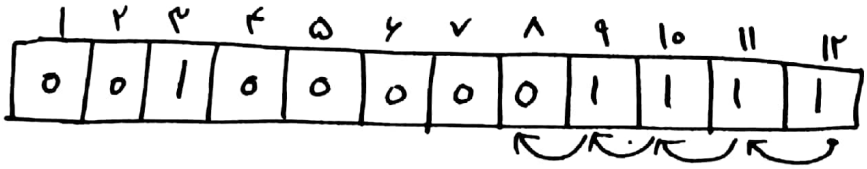
ابتدا با استفاده از پیمایش in-order همه‌ی گره ها را
مرتب می‌کنیم. حالا مجموع همه‌ی گره ها از a تا b
را حساب می‌کنیم. و بر تعداد آنها تقسیم می‌کنیم.

پیمایش in-order از مرتبه‌ی O(n) است.
محاسبه مجموع گره ها از a تا b نیز حداکثر O(n) است.
بنابراین محاسبه‌ی میانگین از مرتبه‌ی O(n) خواهد بود حتی
اگر ارتفاع درخت logn باشد.

۵۲ گزینه (۱)

پاسخ کوتاه و سریع:

① اگر بخواهیم پاسخ کوتاه و سریع به این مسأله بدهیم تصور می‌کنیم که برای آرایه‌ی مانند آرایه‌ی زیر اگر از انتها به ابتدا خانه‌ها را ملاقات کنیم، می‌توانیم فقط با یک بار پیمودن این آرایه یعنی در $\Theta(n)$ همه‌ی مسأله‌ها را پاسخ بدهیم.



مثلاً از $A[12]=1$ شروع می‌کنیم و آنقدر به عقب می‌رویم تا به اولین (تغییر) بدیم. اولین تغییر در اندیس $j=8$ رخ می‌دهد پس جواب مسأله برای $i=12$ می‌شود $j=8$.

تا حالا ۵ خانه را ملاقات کرده‌ای. حالا از خودت بپرس آیا برای یافتن جواب مسأله برای $i=11$ لازم است دوباره از خانه‌ی یازدهم شروع به عقب رفتن کنی؟

معلوم است که لازم نیست و خانه‌ی $j=8$ جواب مشترک همه‌ی خانه‌های ۱۲، ۱۱، ۱۰، ۹ است.

به این ترتیب با یک الگوریتم مناسب، می‌توانیم با یک بار طی کردن این آرایه،

به این ترتیب با یک الگوریتم مناسب، می‌توانیم طعمی مسأله‌ها را برای $i=n$ ، $i=n-1$ ، ... ، $i=1$ فقط یک بار طی کردن این آرایه حل کنیم.

مجموع زمان مورد نیاز برای حل این n مسأله، می‌شود $T(n)=\Theta(n)$ پس با تقسیم این عدد بر تعداد مسأله‌ها داریم:

$$\frac{T(n)}{n} = \frac{\Theta(n)}{n} = \Theta(1) \in O(1)$$

اکنون به پاسخ کامل‌تر و کُد مورد نظر توجه کنید:

② پاسخ تشریحی کامل برای تست ۵۲ :

ابتدا مساله را به صورت مفهومی مورد تحلیل قرار می دهیم. اگر ما فقط یک بار این آرایه را از انتها به ابتدا یعنی از $A[n]$ تا $A[1]$ طی کنیم می توانیم برای همه ی خانه ها، کار مورد نظر را انجام بدهیم.

برای مثال به این آرایه توجه کنید:

(۱)	(۲)	(۳)	(۴)	(۵)	(۶)	(۷)	(۸)	(۹)
۰	۱	۰	۰	۰	۰	۱	۱	۱

(از این مثال به عنوان اثبات استفاده نمی کنیم بلکه به عنوان روشن شدن مطلب در حالت کلی استفاده می کنیم.)

از انتها به ابتدا حرکت می کنیم.

ابتدا $i=9$ است و $j=9$ است.

به عقب می رویم و هر بار از j یک واحد کم می کنیم. البته تا جایی این کار را انجام می دهیم که $A[j]=A[i]$ باشه.

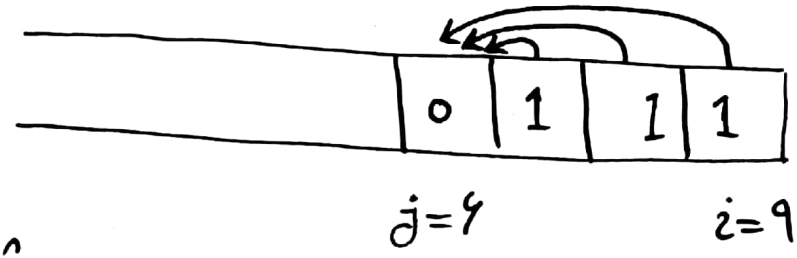
مثلاً $A[9]=A[9]$ است پس: $j=9-1=8$

$A[8]=A[9]$ است پس: $j=8-1=7$

$A[7]=A[9]$ است پس: $j=7-1=6$

حالا می بینیم که $A[6] \neq A[9]$ است.

این به ما نشان می دهد که برای همه خانه های شماره ی ۹, ۸, ۷ جواب مسئله را پیدا کرده ایم.



یعنی اولین جایی که محتوای خانه ها تغییر کرده است، اندیس ششم است. این اندیس اکنون در $j=6$ ذخیره شده است. پس آرایه ی جواب ها یعنی B را به این صورت پُر می کنیم:

```
for K = j+1 to i
    B[k] = j
```

نتیجه آن می شود:

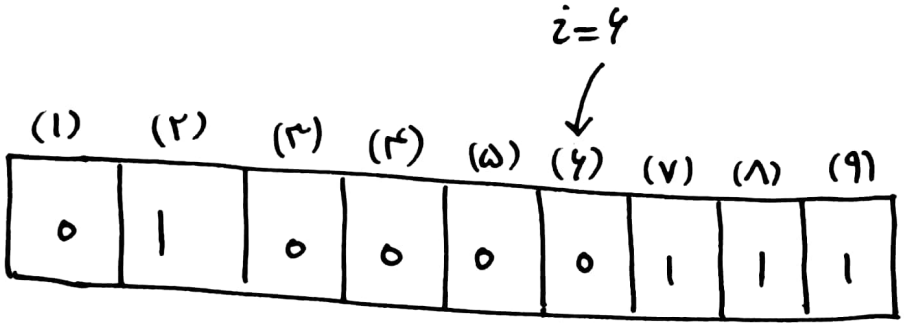
$B[7] = 6$
$B[8] = 6$
$B[9] = 6$

یعنی برای خانه های نهم و هشتم و هفتم، محل جواب می شود خانه ی شماره ی ۶.

حالا باید همین روند را با شروع از خانه‌ی ششم ادامه بدهیم.

دس فرض می‌کنیم: $j=i$ باشد.

یعنی هم اکنون: $i=j=6$ است. و کار را دوباره آغاز می‌کنیم.



در ابتدا $i=6$ و $j=6$ است.

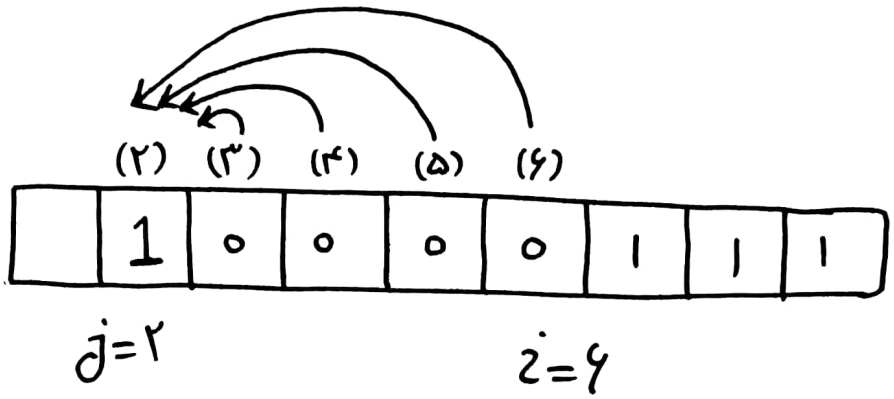
$A[6]=A[6]$ است پس: $j=6-1=5$

$A[5]=A[6]$ است پس: $j=5-1=4$

$A[4]=A[6]$ است پس: $j=4-1=3$

$A[3]=A[6]$ است پس: $j=3-1=2$

حالا دیگر $A[2]\neq A[6]$ است پس اولین باری که رقم‌ها تغییر کردند در خانه‌ی $j=2$ است.



این نشان می‌دهد که برای خانه‌های ۶ و ۵ و ۴ و ۳ جواب ما یعنی اولین تغییر سمت چپ، خانه‌ی دوم است.

```
for  k = j+1  to  i
     B[k] = j
```

در نتیجه:

B[۳] = ۲
B[۴] = ۲
B[۵] = ۲
B[۶] = ۲

حالا باید همین روال را با شروع از خانه‌ی دوم ادامه دهیم

پس فرض می‌کنیم: $j = i = ۲$ باشد.

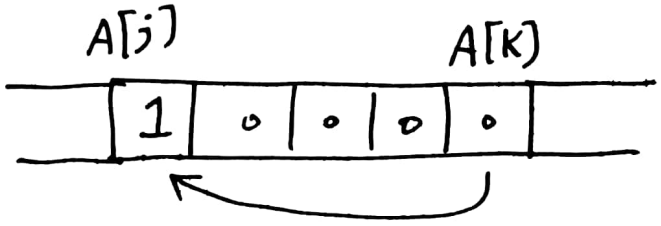
(به همین ترتیب تا انتها ادامه دهید)

توجه: هرچند نیازی به کدنویسی نداریم اما برای کامل‌تر شدن پاسخ به این شبه کد توجه کنید:

ورودی برنامه: آرایه‌ی باینری $A[1,2,\dots,n]$

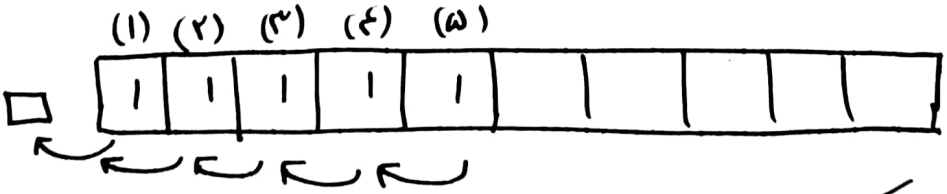
خروجی برنامه: آرایه‌ی $B[1,2,\dots,n]$

می‌گوییم $B[k]=j$ یعنی برای خانه‌ی $A[k]$ می‌گوییم



اولین جایی که در سمت چپ آن، صفر به 1 تبدیل شده یا 1 به صفر تبدیل شده، خانه‌ی با اندیس k است.

توجه: اگر از یک جا به بعد هیچ تغییری مشاهده نشود آنگاه $B[k]=0$ می‌شود که نشانه‌ی آن است که برای اندیس k، جوابی در آرایه‌ی A پیدا نکردیم.



مثلاً در این شکل برای $k=5$ هیچ صفری سمت چپ نداریم پس $B[5]=0$ می‌شود یعنی خانه‌ی صفرم!

شبه کد به زبان ساده :

```
i=n , j=n
while i≥1 do
    while A[j]=A[i] do
        j=j-1
    for k=j+1 to i
        B[k]=j
    i=j
```

وقتی کار تمام می‌شود، جواب مسأله برای خانه‌ی $A[K]$ در $B[K]$ ذخیره شده است. اگر $B[K]=0$ شود یعنی آن خانه، جوابی نداشته است.

A:

(۱)	(۲)	(۳)	(۴)	(۵)	(۶)	(۷)	(۸)	(۹)
۰	۱	۰	۰	۰	۰	۱	۱	۱

B:

(۱)	(۲)	(۳)	(۴)	(۵)	(۶)	(۷)	(۸)	(۹)
۰	۱	۲	۲	۲	۲	۶	۶	۶

نتیجه:

اگر خوب دقت کنید، برای حل کردن این مسئله برای همه‌ی خانه‌های $A[1], \dots, A[n]$ در مجموع، مجبور شدیم هر کدام از خانه‌های $A[j]$ را فقط یک بار ملاقات کنیم و سپس فقط یک بار در B ذخیره کنیم.

پس مجموع هزینه‌ها می‌شود: $T(n) = n + n = 2n$

در هزینه‌ی سرشکن باید این مجموع را بر n تقسیم کنیم؛

$$\text{جواب} = \frac{T(n)}{n} = \frac{2n}{n} = 2 = O(1)$$

(2 ← عدد ثابت)

۵۳ گزینه (۴)

برای حل رابطه‌ی بازگشتی $T(n) = T(\sqrt{n}) + f(n)$

از جایگذاری $n = A^K$ برای $A > 1$ استفاده می‌کنیم. معمولاً $n = 2^K$ را انتخاب می‌کنیم البته انتخاب پایه، دلخواه است.

$$T(n) = T(\sqrt{n}) + O(\log \log n)$$

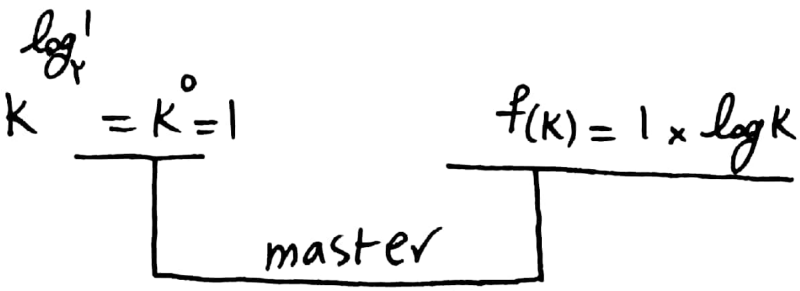
$$T(2^K) = T(2^{\frac{K}{2}}) + O(\log K)$$

توجه کنید که: $\log \log n = \log \log 2^K = \log K$

با نامگذاری $F(K) = T(2^K)$ داریم:

$$F(K) = F(\frac{K}{2}) + O(\log K)$$

از قضیه‌ی اساسی استفاده کنیم:



هردو کفه‌ی ترازو به لحاظ درجه‌ی چند جمله‌ای با هم برابرند.

اما $f(K)$ یک جمله‌ی $\log K$ بیشتر دارد. در این حالت داریم:

$$F(K) = O(f(K) \cdot \log K) = O(\log^2(K))$$

در پایان با توجه به آن که $n = 2^K$ و $K = \log n$ است

داریم:

$$T(n) = O(\log^2(\log n))$$

توجه: اگر $B>1$ یک عدد ثابت باشد، پاسخ رابطه‌ی بازگشتی

$$T(n) = T\left(\frac{\sqrt{n}}{B}\right) + f(n)$$

هیچ تفاوتی به لحاظ مرتبه‌ی رشد، با پاسخ رابطه‌ی بازگشتی:

$$T(n) = T(\sqrt{n}) + f(n)$$

ندارد.

برای مثال، در مسأله‌ی قبل، حتی اگر صورت سؤال چنین بود:

$$T(n) = T\left(\frac{\sqrt{n}}{2}\right) + O(\log\log n)$$

باز هم می‌توانستید فقط رابطه‌ی بازگشتی:

$$T(n) = T(\sqrt{n}) + O(\log\log n)$$

را حل کنید و جواب باز هم $T(n) = O(\log^2(\log n))$ بود.

۵۴ گزینه (۲)

پاسخ کوتاه:

می‌خواهیم آرایه‌ی نامرتب $A[1,2,\ldots,n]$ را مرتب کنیم. تابع یا عملیات SORT(i) هربار که صدا زده شود، یک زیرآرایه به طول K را مرتب می‌کند. (از خانه‌ی $A[i]$ تا $A[i+K-1]$).

اگر فقط بخواهیم تست را پاسخ دهیم، کافیست به حالت بدهی $k=n$ فکر کنیم. اگر $k=n$ باشد یعنی فقط با یک بار صدا زدن SORT(1) همه‌ی خانه‌ها مرتب می‌شوند.

پس در حالت $k=n$ باید جواب ما $O(1)$ باشد.

فقط گزینه (۲) چنین است:

$$\text{if } n=k \Rightarrow \frac{n^2}{k^2}=\frac{n^2}{n^2}=1$$

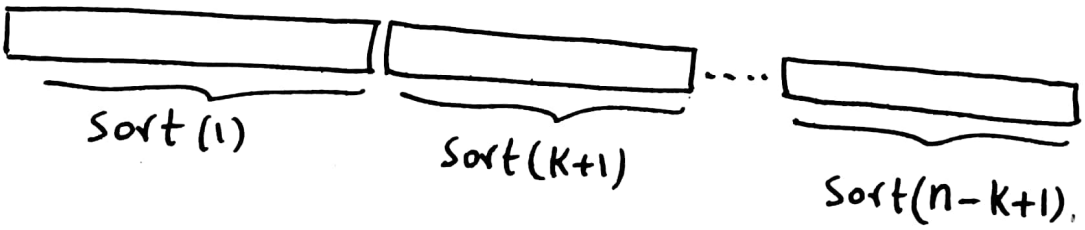
حالا به پاسخ کامل‌تر توجه کنید:

پاسخ تشریحی مبرای تست ۵۴ :

برای راحتی بیشتر ، فرض می‌کنیم که n بر k بخش پذیر باشد. ممکن است راه های مختلفی برای بهره بردن از sort(i) به ذهن ما برسد که باید در اینجا تحلیل شوند.

الف ابتدا توجه کنید که مرتب کردن قسمت های مجزا، که هیچ تداخلی با هم ندارند، نوعی وقت تلف کردن و هدر دادن زمان است.

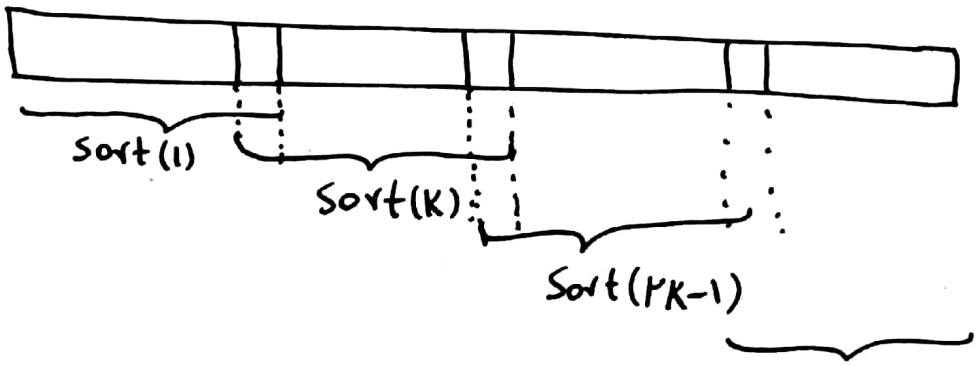
مثلاً اگر شخصی تابع sort(1) و پس sort(k+1) و ... را صدا بزند در پایان $\frac{n}{k}$ تا قطعه دارد که مرتب هستند اما هیچ ربطی به هم ندارند.



برای مثال به ازای $n=9$ و $k=3$ پس از $\frac{n}{k}=3$ مرحله استفاده از sort برای قسمت های مجزا، ممکن است این حالت رخ دهد :

اما این قسمت های مرتب شده، هیچ دردی را دوا نمی‌کنند زیرا هنوز کل آرایه مرتب نشده است. در واقع شما حتی نتوانسته‌اید min را به ابتدای لیست یا max را به انتهای لیست بیاورید در حالی که $\frac{n}{k}$ بار از sort استفاده کرده‌اید.

ب) ایده‌ی دوم آن است که هر بار یک خانه‌ی مشترک را در قطعات به طول K قرار دهیم یعنی به این صورت:



به این ترتیب $\frac{n}{k}$ بار از تابع sort استفاده کرده‌ایم و در پایان مطمئن هستیم که max را به آخرین خانه برده‌ایم.

علت این امر آن است که پس از اجرای اولین sort(1) عددی که در خانه‌ی A[k] قرار دارد از همه‌ی خانه‌های A[1], ..., A[k-1] بزرگتر است.

حالا، پس از اجرای دومین دستور sort(K)،

عددی که در خانه‌ی $A[2K-1]$ قرار دارد از همه‌ی خانه‌های

$$A[K], A[K+1], \ldots, A[2K-2]$$

بزرگ‌تر است. در ضمن $A[K]$ هم از $A[1], \ldots, A[K-1]$ بزرگ‌تر بود

پس می‌توان گفت که قطعاً $A[2K-1]$ از همه‌ی خانه‌های سمت چپ

بزرگ‌تر است. به همین ترتیب، ما همواره اطمینان داریم که آخرین

خانه‌ی sort شده، حاوی max همه‌ی سمت چپی‌ها است.

پس $\frac{n}{K}$ بار عمل sort را انجام می‌دهیم و فقط موفق

می‌شویم که max را به خانه‌ی $A[n]$ ببریم.

این اصلاً خوب نیست. زیرا دفعه‌ی بعد مجبوریم $\frac{n-1}{K}$ بار

عمل sort را تکرار کنیم تا دومین max را به $A[n-1]$

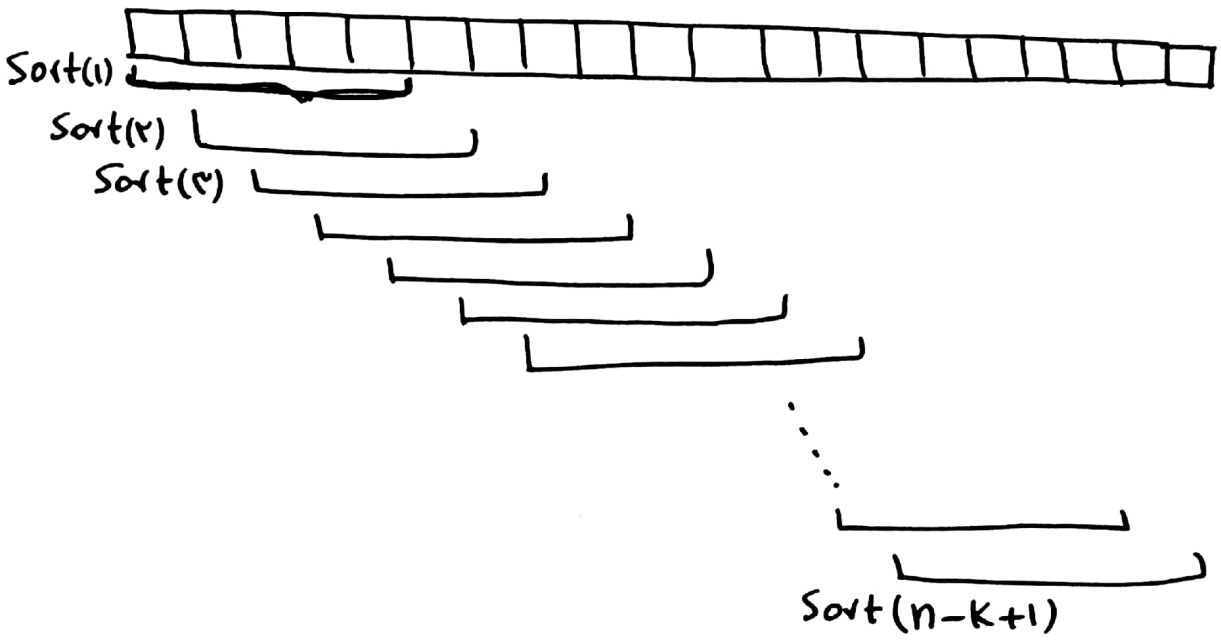
ببریم. به این ترتیب، تعداد کل تکرارهای sort برابر است با:

$$\frac{n}{K} + \frac{n-1}{K} + \frac{n-2}{K} + \cdots + \frac{K}{K} = \frac{1}{K}\left(n + (n-1) + \cdots + K\right)$$

$$\simeq \frac{1}{K}\left(\frac{n^2}{2} - \frac{K^2}{2}\right) = \theta\left(\frac{n^2}{K} - K\right)$$

در ایده‌های بعدی می‌بینیم که مرتبه‌ی بسیار کمتری ممکن می‌شود.

ج: ممکن است به این فکر بیفتید که قطعات مرتب شده را در k-1 خانه با هم مشترک کنید.



خوبی این نوع استفاده آن است که پس از یک دور، استفاده از دستورات sort، شما اطمینان دارید که k-1 خانه‌ی آخر، همان k-1 عنصر بزرگ آرایه هستند. پس دفعه‌ی بعد به جای آرایه‌ای به طول n، باید آرایه‌ای به طول n-(k-1) را در نظر بگیرید.

اما بدی این روش آن است که در همان دور اول تعداد بسیار زیادی دستور sort استفاده کرده‌اید. در شکل می‌بینید که در دور اول، n-k+1 بار از این دستور استفاده کرده‌اید

در دور بعدی، تقریبا $n-2K$ بار از آن استفاده خواهیم کرد.

در دور سوم تقریبا $n-3K$ بار از آن استفاده می‌کنیم.

و مجموع این جملات باز هم $\theta(\frac{n^2}{k})$ می‌شود.

$$T = (n-k) + (n-2k) + (n-3k) + \cdots + (n-(\frac{n}{k})k)$$

اگر فرض کنیم مثلا $m = \frac{n}{k}$ است داریم:

$$T = (m-1)k + (m-2)k + (m-3)k + \cdots + (1)k + 0$$

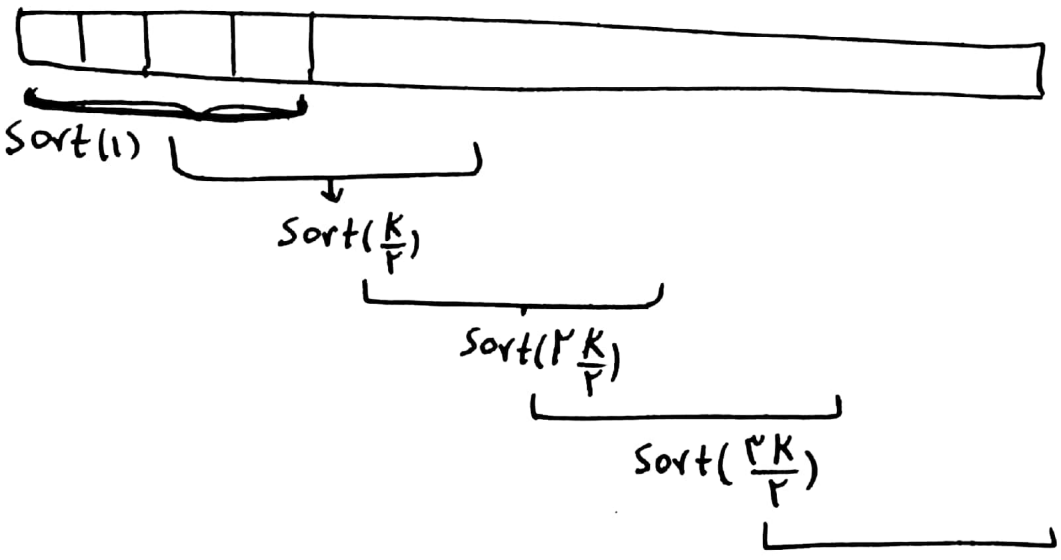
$$= k\left(1 + 2 + \cdots + (m-2) + (m-1)\right)$$

$$\simeq \theta(k\frac{m^2}{2}) = \theta(\frac{n^2}{k})$$

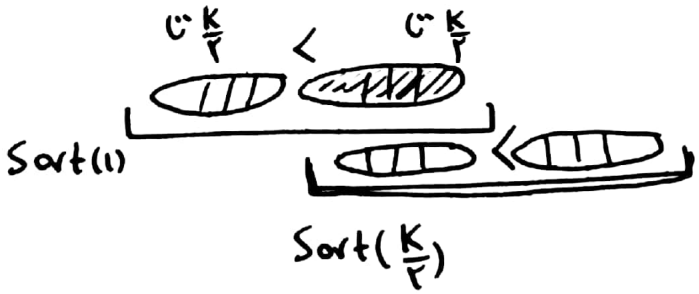
پس باز هم ایده‌ی مناسبی نیست.

د : به قول معروف، خیر الامور، اوسطها.

یعنی نه اشتراک قطعات را تک عضوی بگیریم،
نه اشتراک قطعات را K-1 عضوی بگیریم.
بلکه اشتراک ها را $\frac{K}{2}$ عضوی بگیریم.



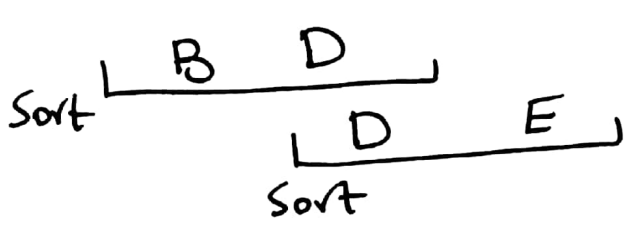
وقتی اولین دستور Sort(1) اجرا می‌شود، نیمه‌ی دوم از نیمه‌ی اول بزرگتر هستند.



وقتی sort دوم را از خانه‌ی $\frac{K}{2}$ تا K خانه‌ی بعدی اجرا می‌کنیم نیمه‌ی دوم از نیمه‌ی اول بزرگتر هستند.

به بیان ساده اگر هر نیمی به طول $\frac{k}{2}$ را با حروف اختیاری

B، D، E، H نشان دهم داریم:



$B < D$

$D < E$

پس بعد از ۲ مرحله، اطمینان داریم که $\frac{k}{2}$ خانه‌ی قرار گرفته در E بزرگترین عددها تاکنون بوده‌اند.

پس:

اولاً: هر دور که این کار را به پایان می‌رسانیم $\frac{k}{2}$ تا از max ها به انتهای آرایه می‌رویم

ثانیاً: چون خانه‌ی شروع sort را هر بار $\frac{k}{2}$ جلوتر می‌بریم

پس در مرحله‌ی اول تعداد sort ها: $\frac{n}{(\frac{k}{2})}$ است.

در مرحله‌ی دوم: $\frac{n-\frac{k}{2}}{\frac{k}{2}}$ ،

در مرحله‌ی سوم: $\frac{n-\frac{k}{2}-\frac{k}{2}}{(\frac{k}{2})}$ ، ....

در نتیجه با کمی دقت داریم:

$$T(n) = \frac{n}{(\frac{K}{2})} + (\frac{n}{(\frac{K}{2})} - 1) + (\frac{n}{(\frac{K}{2})} - 2) + \cdots + 1$$

$$= 1 + 2 + 3 + \cdots + (\frac{n}{\frac{K}{2}} - 1) + \frac{n}{(\frac{K}{2})}$$

$$\simeq \theta\left((\frac{n}{(\frac{K}{2})})^2\right) = \theta(\frac{n^2}{K^2})$$

۵۵ گزینه (۲)

برای سادگی بیشتر فرض کنید key کلید عددی باشد. و $F(key)=$ اندیس انتخاب شده توسط تابع درهم ساز باشد.

فرض یکنواختی به این معناست که احتمال انتخاب همه‌ی کلیدها باهم برابر است با:

به احتمال $\frac{1}{18}$ داریم: $F(key)=1$

به احتمال $\frac{1}{18}$ داریم: $F(key)=2$

⋮

به احتمال $\frac{1}{18}$ داریم: $F(key)=18$

حالا ببینیم واریسی خطی چه تاثیری دارد.

در حال حاضر اندیس های ۱ و ۳ و ۷ و ۹ و ۱۰ و ۱۲ و ۱۴ و ۱۶ و ۱۷ و ۱۸ پر شده اند.

اگر تابع درهم ساز، یکی از این خانه‌ها را انتخاب کند، واریسی خطی به مقدار اندیس انتخاب شده یک واحد اضافه می‌کند تا به یک خانه‌ی خالی برسد. به محض رسیدن به اولین خانه‌ی خالی، کلید key را در آن خانه قرار می‌دهد.

چند مثال برای درک بهتر:

۱) اگر مثلاً $F(key)=14$ انتخاب شود، خانه‌ی چهاردهم قبلاً پر شده است پس وارسی خطی ما را به خانه‌ی بعدی می‌برد و در نهایت کلید key در خانه‌ی پانزدهم قرار خواهد گرفت.

۲) اگر مثلاً $F(key)=9$ انتخاب شود، خانه‌ی نهم پر است پس وارسی خطی ما را به خانه‌ی دهم می‌برد. اما خانه‌ی دهم نیز پُر است پس وارسی خطی ما را به خانه‌ی یازدهم می‌برد و key در خانه‌ی یازدهم قرار خواهد گرفت.

۳) اگر مثلاً $F(key)=5$ انتخاب شود. این خانه هم اکنون خالی است و key به خانه‌ی پنجم می‌رود.

توجه: در وارسی خطی اگر به انتهای لیست رسیدیم و لازم بود جلوتر برویم، به ابتدای لیست برمی‌گردیم در واقع باقی مانده‌ی تقسیم بر ۱۸ را در نظر می‌گیرد. پس مثلاً:

$$18+1=19 \equiv 1$$

هم‌نهشتی به پیمانه‌ی ۱۸

نتیجه:

۱۸ حالت مختلف برای مقدار F(key) داریم که احتمال هرکدام $\frac{1}{18}$ است.

در ۵ تا از این حالت ها، در نهایت key به خانه‌ی دوم می‌رود.

این حالت ها عبارتند از:

F(key) = ۱
F(key) = ۲
F(key) = ۱۶
F(key) = ۱۷
F(key) = ۱۸

پس احتمال آن که در نهایت key به خانه‌ی دوم برود $\frac{5}{18}$ است.

توجه: (یک دیدگاه جالب!)

اگر لیست به طول ۱۸ را به صورت یک دایره تصور کنید،

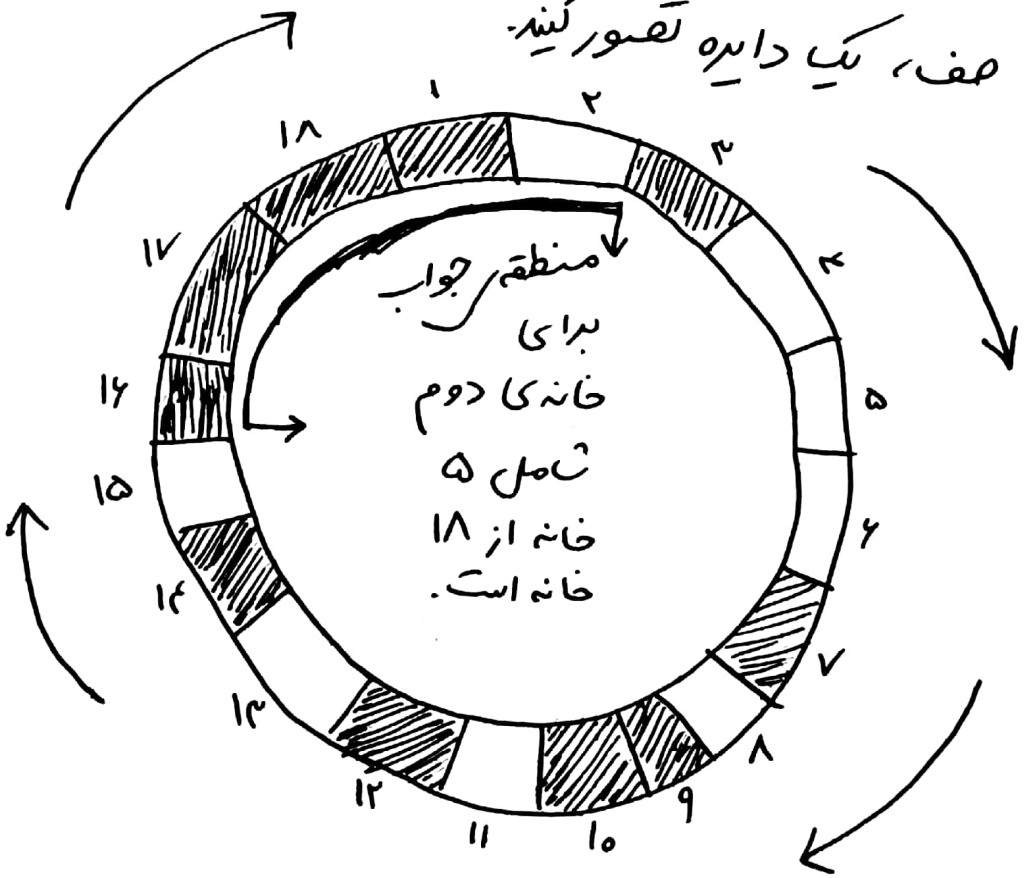
خانه‌ی دوم و همه‌ی خانه‌های دیگر

که قبل از این خانه به صورت متوالی قرار دارند، تعداد حالات

مورد نظر ما هستند.

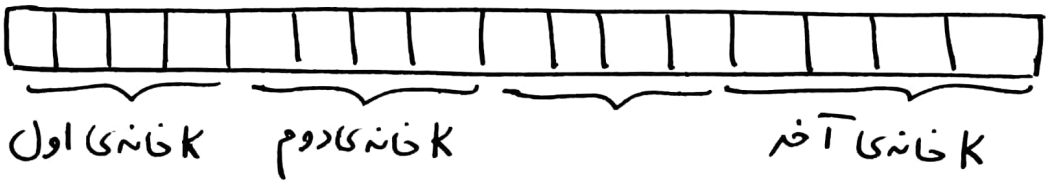
قبلاً هم در گفته بودم که هم‌نشستی یعنی آن که به جای یک

صف، یک دایره تصور کنید.



۵۶ گزینه (۴)

جعبه‌ی سیاه ما هر بار یک زیرمجموعه‌ی حداکثر k عضوی از اندیس‌ها را قبول می‌کند.



ابتدا شماره‌های $1,2,\dots,n$ را به قطعات k عضوی تقلیک کنیم.

$$S_1 = \{1,2,\dots,k\}$$
$$S_2 = \{k+1,\dots,2k\}$$
$$\vdots$$
$$S_{\frac{n}{k}} = \{n-k+1,\dots,n\}$$

تعداد این قطعات برابر است با $\frac{n}{k}$. (البته آخرین قطعه ممکن است کمتر از k عضو داشته باشد اما ایرادی ندارد.)

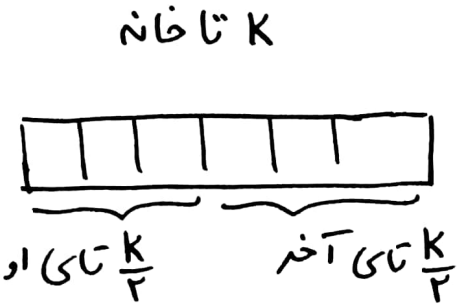
هر کدام از این مجموعه‌ها را به همراه عدد x به جعبه‌ی سیاه می‌دهیم. حال اگر x در مثلاً در قطعه‌ی S_j قرار داشته باشد، متوجه می‌شویم.

تا اینجا $\frac{n}{K}$ بار از جعبه‌ی سیاه استفاده کرده‌ایم.

توجه: ممکن است شانس بیاورید و x در همان قطعه‌ی اول باشد.
ولی روی شانس حساب نکنید. مهم آن است که حداکثر با $\frac{n}{K}$ بار
استفاده از جعبه‌ی سیاه، قطعه‌ی مورد نظر را پیدا می‌کنیم.

مرحله‌ی دوم:

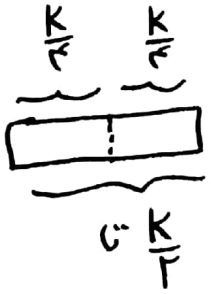
S_{ij}:



می‌دانیم که x در یکی از این K خانه قرار دارد.
آن‌ها را به دو مجموعه‌ی مساوی با اندازه $\frac{K}{2}$ تفکیک می‌کنیم.
نیمه‌ی سمت چپ را به جعبه‌ی سیاه می‌دهیم.
نیمه‌ی سمت راست را به جعبه‌ی سیاه می‌دهیم.
به این ترتیب متوجه می‌شویم که x در کدام نیمه قرار دارد.
تا اینجا $\frac{n}{K}$ بار و پس ۲ بار از جعبه‌سیاه استفاده کرده‌ایم.

به عملیات نصف کردن آن

بخش ادامه می دهیم.



هربار که تعداد خانه‌ها نصف می‌شود، ما دوبار از جعبه سیاه استفاده می‌کنیم تا متوجه شویم x در کدام نیمه قرار دارد.

سپس آن نیمه را نصف می‌کنیم و ادامه می‌دهیم. همان طور که می‌دانید تعداد دفعات نصف کردنِ K تا رسیدن به 1 خانه برابر با $\log K$ است.

نتیجه:

$$\text{تعداد دفعات استفاده از جعبه سیاه} = \frac{n}{K} + \underbrace{(2+2+2+\cdots+2)}_{\text{به تعداد } \log K}$$

$$= \frac{n}{K} + 2\log K$$

$$= O\left(\frac{n}{K} + \log K\right)$$

دقت کنید: در چه حالتی، جواب $O(\log n)$ می‌شد؟

اگر این محدودیت را نداشتیم را که مجموعه‌ی که حداکثر K عضوی باشد،

آنگاه می‌توانستیم از همان ابتدا با نصف کردن اندیس‌های $\{1, 2, \ldots, n\}$ شروع کنیم.

یعنی از ابتدا با مجموعه‌های:

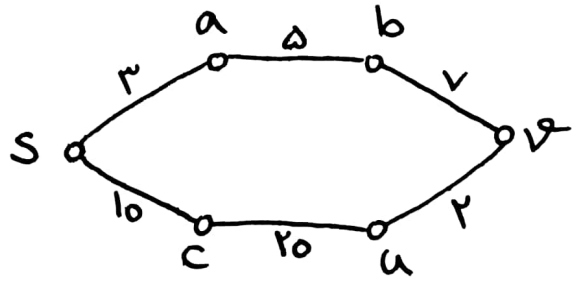
$S_1 = \{1, 2, \ldots, \frac{n}{2}\}$ $S_2 = \{\frac{n}{2}+1, \ldots, n\}$

شروع می‌کردیم. وقتی متوجه می‌شدیم x در کدام نیمه قرار دارد، آن را نصف می‌کردیم.

در این صورت $O(\log n)$ به دست می‌آمد و گزینه (۳) جواب می‌شد.

۵۷ گزینه (۴)

ابتدا نشان می‌دهیم که دکسترا در این مساله درست کار نمی‌کند. پس کافیت یک مثال نقض بیاوریم.



در گراف مقابل گره S را به عنوان مبدأ لحاظ کنید.

الگوریتم‌های دکسترا و بلمن فورد هردو تک منبع هستند یعنی هدف آنها یافتن کوتاهترین مسیرها از گره S به سایر گره‌ها است.

در اینجا از همان ابتدا می‌دانیم که طبق تعریف تازه‌ای که در صورت سئوال آمده است، وزن مسیر SCDF برابر است با ۲ و وزن مسیر SABF برابر است با ۳.

زیرا در این سئوال چنین فرض کرده است که به جای مجموع وزن یال‌های مسیر، فقط وزن سبک‌ترین یال آن مسیر را به عنوان وزن مسیر در نظر می‌گیریم.

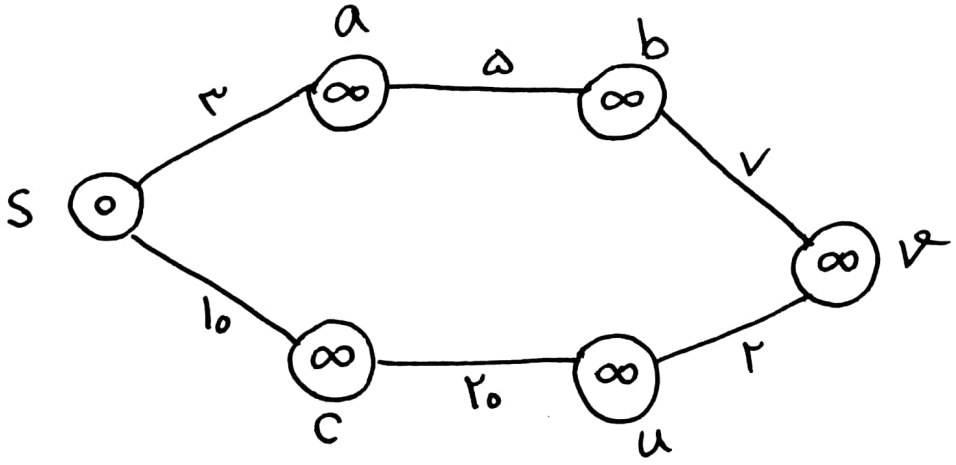
با این حساب، در این گراف، مسیر SCDF سبک‌ترین

مسیر از s به v است و اگر دکسترا درست عمل کند باید در پایان $d[v]=2$ را به ما بدهد.

حالا می بینیم که چنین نیست.

ابتدا مقادیر اولیه را به گره ها می دهیم:

طبق مقادیر اولیه‌ی استاندارد در دکسترا و یمن فورد، باید $d[s]=0$ و در مورد سایر گره ها $d(u)=\infty$ را فرض کنیم.



در این مرحله:

$S=\{s\}$ $\quad V-S=\{a,b,v,c,u\}$

حالا چون $s\in S$ است، فقط گره های مجاور s را آپدیت (به روزرسانی) می کنیم

در الگوریتم معمولی دکسترا می‌گوییم:

if $d(a) > d(s) + w(s,a)$ then

$$d(a) = d(s) + w(s,a)$$

اما در این سئوال به جای $+$ از $\min$ استفاده می‌کنیم:

$$d(a) = \min(d(s), w(s,a))$$
$$= \min(0, 3) = 0$$

همچنین:

$$d(c) = \min(d(s), w(s,c))$$
$$= \min(0, 10) = 0$$

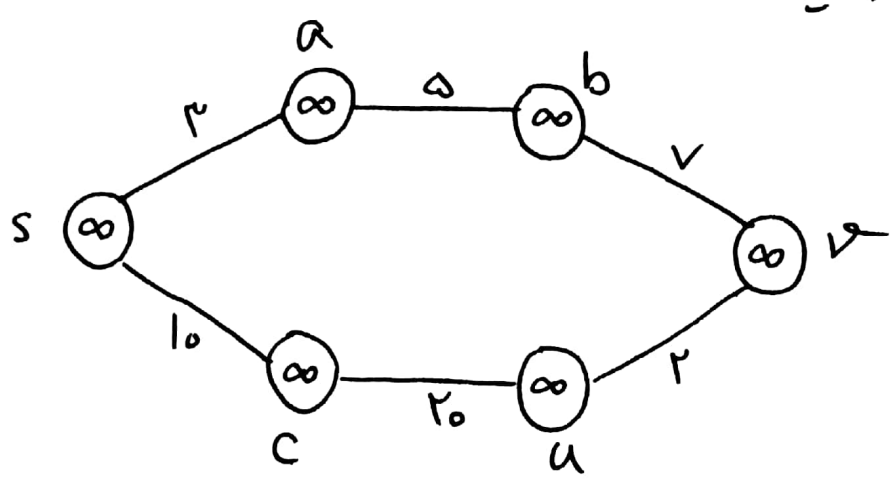
اگر به همین منوال ادامه دهیم، در مراحل بعدی، این عدد صفر به همه‌ی گره‌ها سرایت می‌کند. درحالی که باید $d(v) = 2$ به دست می‌آمد.

پس بدیهی است که با مقداردهی اولیه‌ی استاندارد، به جواب مورد نظر نخواهیم رسید.

بررسی با رفع مشکل مقادیر اولیه :

فرض کنید مشکل مقداردهی اولیه را به این صورت حل کنیم که برخلاف روال معمول، مقدار $d(s)$ را هم ∞ بگیریم.

پس به این صورت آغاز می‌کنیم:



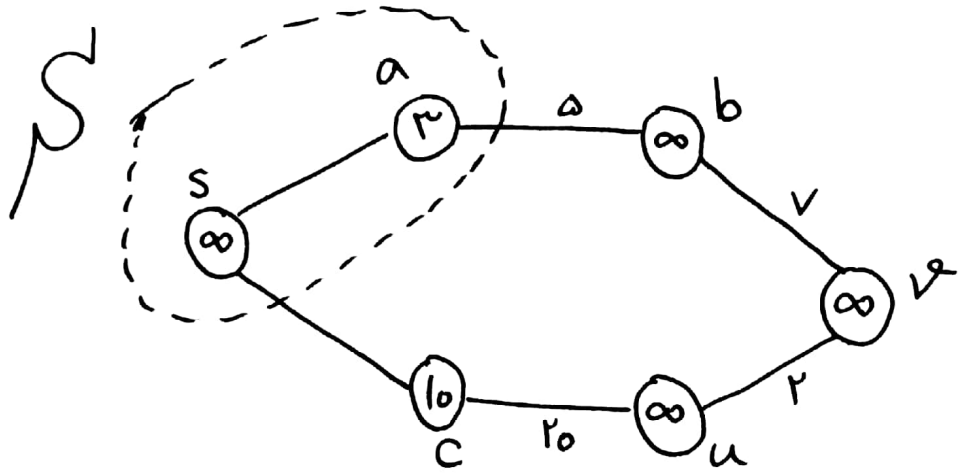
همچنین این نوع مقداردهی اولیه، از همان ابتدا با روال الگوریتم دکسترا و بلمن فورد در تضاد است اما فرض کنیم همان طور که طراح سئوال خواسته، همه‌ی مقادیر اولیه ∞ باشند و ما به طریقی به الگوریتم بفهمانیم که در ابتدا از s شروع کند. پس:

$$S = \{s\} \qquad V - S = \{a, b, v, u, c\}$$

در گام اول، گره های مجاور s آپدیت می شوند:

$$d(a) = \min(\infty, ۳) = ۳$$
$$d(c) = \min(\infty, ۱۰) = ۱۰$$

حالا این a هست که کمترین وزن را در بین گره های $V-S$ دارد. این گره را به مجموعه‌ی S اضافه می کنیم:



در حال حاضر: $S = \{s, a\}$

و چون a عضو تازه وارد به S است باید گره های مجاور a که در خارج از S قرار دارند به‌روزرسانی شوند.

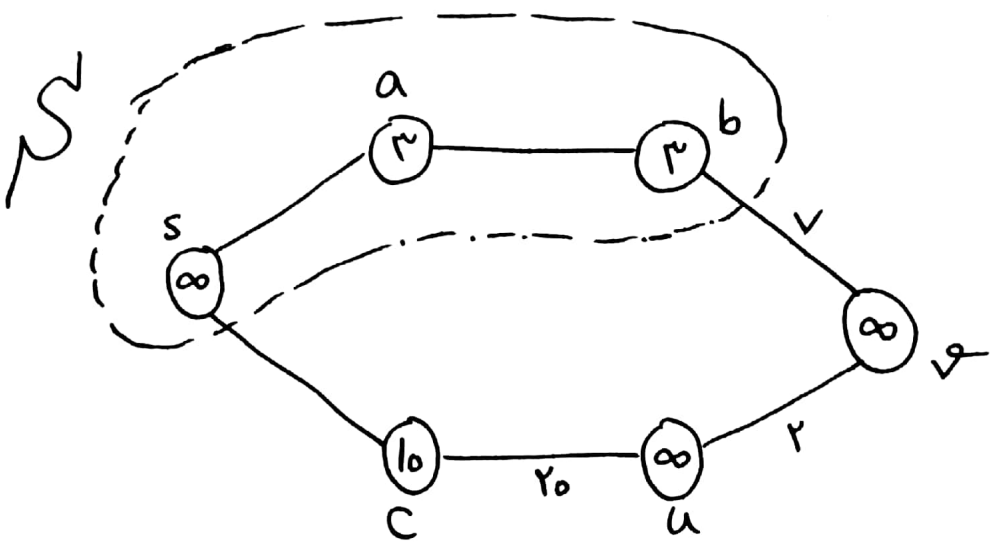
پس:

$$d(b) = \min(d(a), w(a,b))$$
$$= \min(۳, ۵)$$
$$= ۳$$

حالا بین همه‌ی گره‌های b, c, d, u این گره b

است که کمترین وزن را دارد: $d(b) = 3$ پس باید b

را به گ اضافه کنیم:



$گ = \{ s, a, b \}$

حالا b عضو تازه وارد گ است و باید گره‌های مجاور

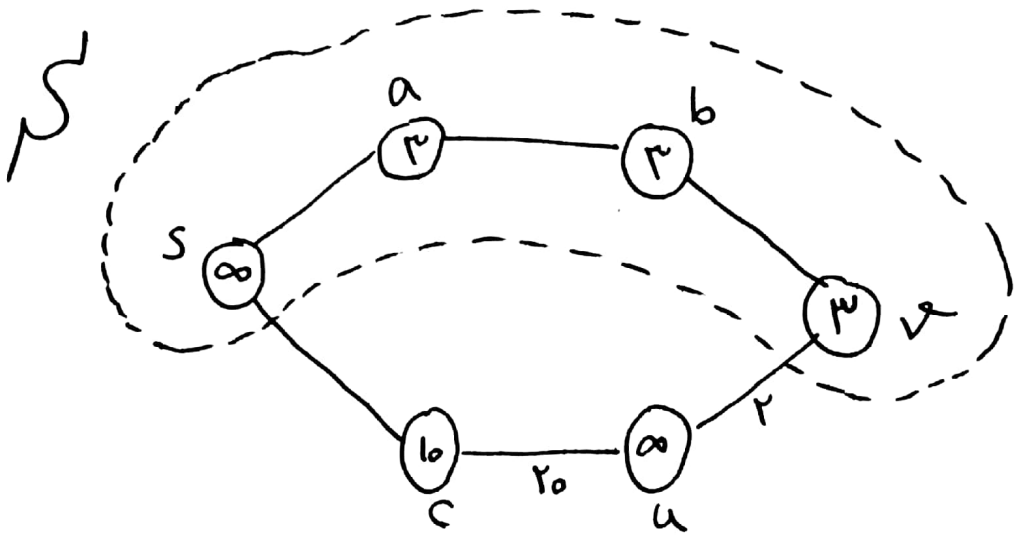
با b را که خارج از گ قرار دارند به روز رسانی کنیم:

$$d(d) = \min(d(b), w(b,d))$$
$$= \min(3, 7)$$
$$= 3$$

حالا بین c, u, v این v است که کمترین وزن

را دارد: $d(v) = 3$

پس آن را به S اضافه می‌کنیم:



در همین مرحله معلوم است که دکتر ا به جواب صحیح نرسیده است

زیرا دیگر مقدار $d(v) = 3$ تغییر نخواهد کرد.

با ادامه‌ی این روند، در مرحله‌ی بعد، $d(u) = 2$

و در مرحله‌ی آخر $d(c) = 2$ می‌شود.

این در حالی است که ما از ابتدا می‌دانستیم باید $d(v) = 2$

به دست آید.

نتیجه:

مهم‌ترین ایراد در این است که در دکسترا و در بلمن فورد مقدار اولیه برای گره منبع $d(s)=0$ است و اگر طبق صورت سؤال، وزن یال‌ها مثبت باشد، معلوم است که:

$$d(a) = \min(d(s), w(s,a))$$
$$= \min(0, w(s,a))$$
$$= 0$$

و این مقدار صفر، مدام به همه گره‌ها گسترش می‌یابد درحالی که مثلاً جواب درست برای گره a باید $d(a)=2$ باشد چون مسیر scuba دارای وزن ۲ است.

ایراد دوم آن است که حتی اگر $d(s)=\infty$ را فرض کنیم، انتخاب اولین یال‌ها و اولین گره‌ها ممکن است ما را از مسیر بهینه دور کند. (مطابق مثال)

[برای بلمن فورد نیز مثال مناسبی نوشته و به‌عنوان تمرین تحلیل کنید].

۵۸ گزینه (۱)

مسئله‌ی بسیار ساده‌ای است.

توصیه می‌کنم جزوه‌ی کامل MST را در کانال
@abolfazlgilak

مطالعه بفرمایید.

الف: اگر از هر دور، سنگین‌ترین یال (یا یکی از سنگین‌ترین یال‌ها) را حذف کنیم قطعاً به درخت MST می‌رسیم. البته ایراد الف آن است که هر بار باید برای یافتن سنگین‌ترین یال دوری که ایجاد شده است، زمان صرف کند (حداکثر $O(n)$).
با این حال حتماً به MST خواهد رسید. پس (الف) صحیح است اگرچه روش کارآمدی نیست.

ب: این روش نیز حتماً به MST می‌رسد و تقریباً مشابه Prim است. البته در Prim ما اصلاً اجازه نمی‌دهیم دور ایجاد شود که بخواهیم آن را حذف کنیم. اما به هر حال در این روش، هنگامی که با اضافه کردن e_k دور ایجاد می‌شود، می‌دانیم که سنگین‌ترین یال دور همین e_k است پس نسبت به (الف) روش سریع‌تری است. به هر حال هر دو گزاره‌ی الف و ب صحیح هستند.

۵۹ گزینه (۳)

سومین رابطه بازگشتی به وضوح صحیح است. در واقع منطق آن به این صورت است:

یا x_n در بزرگترین زیردنباله‌ی صعودی، حضور دارد یا حضور ندارد. اگر حضور نداشته باشد: $LIS(X_{n-1})$ جواب است. اگر حضور داشته باشد، آنگاه ضمن آن که x_n به عنوان آخرین عضو این زیردنباله چاپ می‌شود، عناصر قبلی هم همگی باید از x_n کوچکتر باشند. برای مثال: در رشته‌ی ۱۴۲۵۳۶

در دنباله‌ی X_n: $1, 7, 3, 8, 4, 5, 6$

بزرگترین زیردنباله‌ی صعودی به صورت $1,3,4,5,6$ است.

اگر $x_n=6$ را کنار بگذاریم، سایر عناصر عبارتند از بزرگترین زیردنباله‌ی صعودی X_{n-1} با شرط کمتر بودن از ۶.

$$LIS(X_{n-1}, \overset{x_n}{6}) = 1,3,4,5$$

که سپس در پایان آنها x_n هم قرار می‌گیرد:

$$\langle LIS(X_{n-1}, \overset{x_n}{6}), \overset{x_n}{6} \rangle = 1,3,4,5,6$$

با این توضیحات می‌بینیم که سومین رابطه بازگشتی، صحیح است.

از طرفی، اولین رابطه بازگشتی، درواقع نتیجه‌ی سومی است.

برای راحتی بیشتر فرض می‌کنیم که:

$$S_n = \langle LIS(X_{n-1}, x_n) , x_n \rangle$$

باشه. طبق رابطه‌ی سوم داریم:

۱) $$LIS(X_n) = \max\left[LIS(X_{n-1}) , S_n \right]$$

حالا طبق همین رابطه بازگشتی داریم:

۲) $$LIS(X_{n-1}) = \max\left[LIS(X_{n-2}) , S_{n-1} \right]$$

رابطه‌ی (۲) را در اولی قرار دهیم:

$$LIS(X_n) = \max\left[LIS(X_{n-2}) , S_{n-1} , S_n \right]$$

با تکرار این روند خواهیم داشت:

$$LIS(X_n) = \max\left[S_1, S_2, \dots, S_n \right]$$

$$= \max_{1 \le i \le n} \left[S_i \right]$$

یعنی همان رابطه‌ی بازگشتی اول به دست می‌آید:

$$LIS(X_n) = \max_{1 \le i \le n} \langle LIS(X_{i-1}, x_i) , x_i \rangle$$

تا اینجا دیدیم که سومی و اولی صحیح هستند. (درواقع اولی، نتیجه‌ی سومی است).

اما رابطه‌ی دوم نادرست است. زیرا فقط حالتی را در نظر گرفته که x_n در بزرگترین زیردنباله صعودی حضور دارد. برای مثال فرض کنید:

X_n : ۱۳۴۵۶۷۸۲ ↪ x_n

دراین صورت بزرگترین زیردنباله‌ی صعودی برابر است با:

۱۳۴۵۶۷۸

درحالی که اگر بخواهیم حتماً x_n حضور داشته باشد آنگاه:

$LIS(X_{n-1}, x_n) = 12$

پس به وضوح رابطه‌ی دوم نادرست است.

توضیح کامل ک:

به این دنباله توجه کنید:

	x_1	x_2	x_3	x_4	x_5	x_6
X:	۱	۸	۲	۶	۹	۳

از همین حالا می دانیم که بزرگترین زیردنباله صعودی، ۱۲۶۹ است که طول آن برابر با ۴ است.

حالا به این مقادیر توجه کنید:

۱) اگر $x_1=1$ را به عنوان آخرین عضو زیردنباله درنظر گیریم:

$$(LIS\langle X_0 \, , x_1\rangle \, , x_1) = 1$$

(LIS از قبلی‌ها در انتخاب شود)

۲) اگر $x_2=8$ را به عنوان آخرین عضو زیردنباله درنظر گیریم،

$$(LIS\langle X_1 \, , x_2\rangle \, , x_2) = ۱۸$$

۳) اگر $x_3=2$ را به عنوان آخرین عضو زیردنباله درنظر گیریم:

$$(LIS\langle X_2 \, , x_3\rangle \, , x_3) = ۱۲$$

دقت کنید برای صعودی بودن زیردنباله، دیگر ۸ نمی‌تواند به کار رود.

۴) اگر $x_4 = 6$ را به عنوان سقف و آخرین عدد در نظر گیریم:

$$(LIS < X_3, x_4 >, x_4) = 1\ 2\ 6$$

۵) اگر $x_5 = 9$ را به عنوان سقف و آخرین عدد در نظر گیریم:

$$(LIS < X_4, x_5 >, x_5) = 1\ 2\ 4\ 9$$

$LIS < X_4, x_5 >$: از قبلی‌ها با در نظر گرفتن سقفِ x_5، LIS بگیر.

x_5: x_5 را جواب بگیر

۶) اگر $x_6 = 3$ را به عنوان سقف و آخرین عدد در نظر گیریم:

$$(LIS < X_5, x_6 >, x_6) = 1\ 2\ 3$$

واضح است که یکی از همین‌ها بزرگترین زیردنباله‌ی صعودی خواهد بود. کافیست از بین این ۶ حالت، آن را که max طول را دارد انتخاب کنیم.

پس گزاره‌ی اول منطق ساده و واضحی دارد.

۶۰ گزینه (۳)

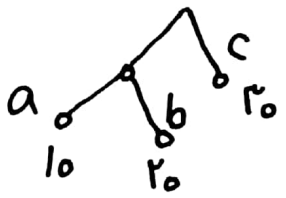
واضح است که این الگوریتم درخت بهینه را برای $n=1$ و $n=2$ تولید می‌کند.

برای مثال برای $n=3$ هم این الگوریتم درخت بهینه را می‌دهد. فرض کنید حروف a، b، c را داریم که فراوانی آنها به صورت زیر است:

f:	a	b	c
	۱۰	۲۰	۳۰

(در این مثال اگر به جای عددهای ۱۰،۲۰،۳۰ حالت کلی $f_1 < f_2 < f_3$ قرار دهید فرقی ندارد)

در درخت بهینه‌ی هافمن، داریم:



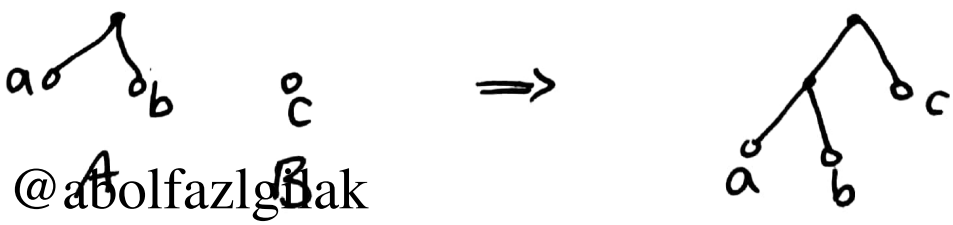
هزینه (تعداد بیت‌ها) $= ۳۰ + ۲\times۱۰ + ۲\times۲۰ = ۹۰$

حالا اگر از الگوریتم داده شده در صورت سئوال استفاده کنیم باید فرض کنیم $A=\{a,b\}$ و $B=\{c\}$ تا مجموعه فراوانی هر دو مجموعه، تا حد ممکن یکسان شوند:

f_A: $۱۰+۲۰$

f_B: ۳۰

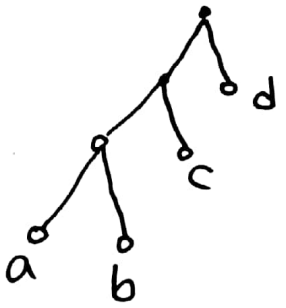
حالا برای هرکدام از آنها درخت هافمن جداگانه تشکیل داده و سپس آنها را به یک ریشه متصل می‌کنیم:



می بینیم که همان درخت بهینه هافمن به دست می آید.
اکنون نشان می دهیم که برای $n=4$ حرف، این الگوریتم، بهینه نیست. مثلاً
فرض کنید این حروف را با این فراوانی ها داریم:

	a	b	c	d
f:	۱۰	۲۰	۳۰	۴۰

اگر خودمان از درخت هافمن استفاده کنیم خواهیم داشت:



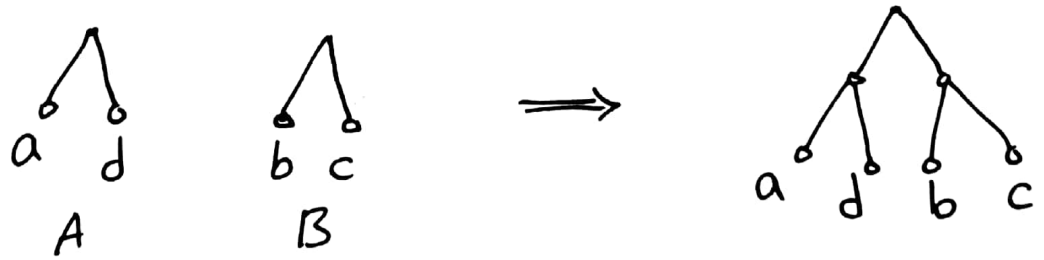
هزینه $= ۴۰ + ۲(۳۰) + ۳(۲۰) + ۳(۱۰) = ۱۹۰$

حالا ببینیم الگوریتم پیشنهاد شده در صورت سئوال چه می کند؟

$A = \{a, d\}$ $\quad B = \{b, c\}$

تفکیک حروف به A و B باید به این صورت باشد تا فراوانی ها حتی الامکان
باهم برابر شوند. $f_A = ۱۰ + ۴۰$ $\quad f_B = ۲۰ + ۳۰$

حالا درخت هافمن هرکدام را جداگانه تشکیل داده و سپس به یک ریشه
متصل کنیم:



هزینه ی درخت ایجاد شده بسیار بیشتر از حالت بهینه است.

هزینه $= ۲(۱۰ + ۲۰ + ۳۰ + ۴۰) = ۲۰۰$

پس به ازای $n=4$ الگوریتم جواب بهینه را نمی دهد.