به نام خدا

طراحی الگوریتم

نمونه‌ی امتحان میان‌ترم

مدت امتحان: ۲ ساعت

توجه

- در هر سوال، پس از طراحی الگوریتم و توضیح آن، شبه‌کد (pseudocode) مربوطه را نوشته و تحلیل زمان اجرا و فضای حافظه‌ی آن‌را بدست آورید.
- توصیه می‌شود قبل از خواندن پاسخ‌ها، سعی کنید سوال را خودتان حل نمایید.

۱. (۲۰ نمره) فرض کنید در الگوریتم quicksort، هر بار آرایه به نسبت α و $1-\alpha$ تقسیم می‌شود که α یک ثابت است و $0 < \alpha \leq \frac{1}{2}$. نشان دهید که حداقل عمق یک برگ در درخت بازگشتی تقریبا $-\log n / \log \alpha$ و حداکثر عمق تقریبا $-\log n / \log(1-\alpha)$ است. (نگران خطای گرد کردن اعداد صحیح در محاسبات نباشید.)

پاسخ: (۱۰ نمره اثبات مقدار کمینه و ۱۰ نمره اثبات مقدار بیشینه) اگر درخت اجرای بازگشتی را رسم کنید، کمترین عمق متعلق به قسمتی است که هر بار بخش کوچکتر آرایه به آن اختصاص پیدا می‌کند (به عبارتی، بخشی که ضریب α دارد.) هربار الگوریتم، اندازه‌ی آرایه را از n به αn کاهش می‌دهد. پس از i مرحله (در عمق i)، اندازه به $\alpha^i n$ کاهش می‌یابد. هنگامی به برگ می‌رسیم که آنقدر آرایه شکسته شده باشد که طول آن یک باشد. به عبارتی، $\alpha^m n = 1$. این عبارت را می‌توانیم به صورت $\alpha^m = \frac{1}{n}$ نیز بنویسیم. با حل این رابطه (با گرفتن لگاریتم از طرفین) خواهیم داشت:

$$m \log \alpha = -\log n \Rightarrow m = -\log n / \log \alpha$$

به طور مشابه، بیشترین عمق، مربوط به بخشی است که هربار بخش بزرگتر آرایه به آن اختصاص پیدا می‌کند (به عبارتی، بخشی که ضریب $1-\alpha$ دارد.) در عمق M هنگامی به آرایه‌ای با یک عنصر می‌رسیم که داشته باشیم $n(1-\alpha)^M = 1$. با حل این عبارت می‌بینیم که حداکثر عمق $M = -\log n / \log(1-\alpha)$ خواهد بود.

دقت کنید که این مقادیر تقریبی است. چون در هر مرحله، آرایه دقیقا به α و $1-\alpha$ تقسیم نمی‌گردد.

---

۲. (۲۵ نمره) یک آرایه مرتب شده به طول n در حافظه خوانده شده که به اندازه k بار به راست شیفت چرخشی داده شده است. مثلا آرایه‌ی [۱۲، ۹، ۵، ۳، ۱] اگر به اندازه $k=2$ شیفت چرخشی به راست داده شود، به آرایه [۵، ۳، ۱ ، ۱۲ ، ۹] تبدیل می‌گردد. الگوریتمی بیابید که میزان k (یا تعداد شیفت‌ها) را در کمترین زمان پیدا کند.

پاسخ: (۱۰ نمره توضیح الگوریتم و حالت‌بندی درست برای رابطه بازگشتی، ۱۰نمره شبه کد، ۵ نمره تحلیل زمان و حافظه؛ راه‌حل خطی، فقط ۳ نمره دارد.) این مسئله، معادل یافتن اندیس عنصر کمینه است (با فرض اینکه آرایه از اندیس صفر شروع شود.) راه‌حل بدیهی، بررسی همه‌ی عناصر آرایه است که پیچیدگی زمان اجرای $\mathcal{O}(\mathrm{n})$ دارد. الگوریتم بهتر، استفاده از روشی مشابه جستجوی دودویی است که پیچیدگی زمان اجرای $\mathcal{O}(\log \mathrm{n})$ دارد. جزئیات این روش بدین صورت است. کمترین عنصر آرایه، تنها عنصری است که عنصر قبلی آرایه از آن بزرگ‌تر است. اگر عنصری قبل آن نبود، آرایه شیفت نخورده یا تعداد شیفت‌ها ضریبی از n است. در ابتدا عنصر وسط با دو عنصر کناری ($\mathrm{mid}+1$ و $\mathrm{mid}-1$) مقایسه می‌شوند:

- اگر عنصر وسط کوچکتر از عنصر $\mathrm{mid}-1$ باشد، عنصر وسط عنصر کمینه است.
- اگر عنصر وسط بزرگتر از عنصر $\mathrm{mid}+1$ باشد، عنصر $\mathrm{mid}+1$ عنصر کمینه است.

در غیر این صورت، عنصر کمینه یا در نیمه‌ی راست یا در نیمه چپ قرار دارد:

- اگر عنصر وسط از آخرین عنصر کوچک‌تر باشد، عنصر کمینه در نیمه چپ قرار دارد.
- در غیر این صورت، در نیمه‌ی راست قرار دارد.

با این روش، آرایه هربار به نصف شکسته شده تا کمترین عنصر پیدا شود. شبه کد مربوطه به صورت زیر می‌باشد:

```
FindRotationCount(A, left, right) {
    if left = right
        return left
    mid = (left + right) / 2
    if mid > 0 AND A[mid] < A[mid - 1]
        return mid

    if mid < right AND A[mid] > A[mid + 1]
        return mid + 1
    // Right array is sorted, search in the left array
    if A[mid] <= A[right]
        return FindRotationCount(A, left, mid-1)

    // Left array is sorted, search in the right array
    return FindRotationCount(A, mid+1, right)
}
```

زمان اجرا $O(\log n)$ و میزان حافظه مصرفی $\mathcal{O}(\mathrm{n})$ و حافظه کمکی $\mathcal{O}(1)$ می‌باشد.

---

۳. (۴۰ نمره) فرض کنید یک فایل متنی به صورت یک رشته با n حرف به شما داده شده است که تمام کلمات آن به هم چسبیده‌اند. به عبارتی فاصله‌ای بین کلمات وجود ندارد. شما می‌خواهید طوری فاصله بین کلمات درج کنید که متن حاصل با معنی و خوانا شود. به عنوان مثال، اگر در ورودی عبارت «منمشتعلعشقعلیمچهکنم» را دریافت کردید، رشته‌ی فاصله‌گذاری شده به صورت «من مشتعل عشق علیم چه کنم» خواهد بود. برای کمک به شما، تابعی در اختیارتان قرار گرفته به نام $f(w)$ که به ازای هرکلمه (مجموعه‌ای از حروف) نشان‌دهنده‌ی میزان با معنی بودن آن است. هدف شما یافتن الگوریتم با زمان اجرای $\mathcal{O}(n^2)$ است که مجموع $f(w)$ را روی همه کلمات جدا شده متن به حداکثر برساند. نیازی به چاپ رشته‌ی فاصله‌گذاری شده نمی‌باشد.

پاسخ: (۱۰ نمره توضیح الگوریتم، ۱۰ نمره رابطه بازگشتی صحیح، ۱۵ نمره شبه کد، ۵ نمره تحلیل حافظه) فرض کنید پاسخ بهینه $x_1x_2x_3...x_k$ باشد که متن را به k کلمه‌ی x_1، x_2 الی x_k شکسته است. بر همین اساس، $x_1x_2x_3...x_{k-1}$ پاسخ بهینه است اگر کلمه‌ی x_k را از متن حذف کنیم، زیرا اگر پاسخ بهینه

نباشد، می‌توان پاسخ دیگری پیدا کرد تا بر آن اساس پاسخی بهتر از $x_1x_2x_3...x_k$ بدست آورد. حال فرض کنید $Opt(i)$ برابر بیشینه‌ی مجموع $f(w)$ روی i حرف اول است. با توجه به ساختار بهینه‌ی زیرمسئله‌ها می‌توان رابطه‌ی بازگشتی زیر را نوشت:

$$Opt(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max\limits_{1 \le j \le i} \{Opt(j-1) + f(j \dots i)\} & \text{otherwise} \end{cases}$$

در این رابطه، $f(j \dots i)$ مقدار تابع را برای حروف اندیس j الی i محاسبه می‌کند. پاسخ مطلوب مسئله، $\mathrm{Opt}(n)$ می‌باشد که به کمک برنامه‌ریزی پویا می‌توان آن‌را محاسبه کرد. به دلیل وجود دو حلقه‌ی تو در تو، زمان اجرا $\mathcal{O}(n^2)$ خواهد بود. پیچیدگی حافظه، $\mathcal{O}(\mathrm{n})$ می‌باشد.

```
WordSegmentation(X) {
    // Array indices are zero based.
    opt[0] = 0
    n = X.length
    for i = 1 to n {
        opt[i] = - ∞
        for j = 1 to i {
            opt[i] = max(opt[i], opt[j-1] + f(X[j-1 ... i-1]))
        }
    }
    return opt[n]
}
```

---

۴. (۲۵ نمره) آرایه‌ی $A=\{a_1, a_2, ..., a_n\}$ و عدد s به شما داده شده است. الگوریتمی به کمک برنامه‌ریزی پویا با زمان اجرای غیر نمایی (شبه چندجمله‌ای) طراحی کنید که مشخص کند آیا می‌توان زیرمجموعه‌ای از A را پیدا کرد که مجموع آن‌ها برابر s شود.

پاسخ: (۱۰ نمره توضیح الگوریتم و رابطه بازگشتی صحیح، ۱۰ نمره شبه کد، ۵ نمره تحلیل زمان و حافظه) این سوال مشابه مسئله کوله‌پشتی صفر-یک می‌باشد، با این تفاوت که ارزش اشیا برابر است و می‌خواهیم بدانیم آیا می‌توانیم کوله‌پشتی را به طور کامل پر کنیم یا خیر.

فرض کنید S(i,j) نشان‌دهنده‌ی این است که آیا می‌توان زیرمجموعه‌ای از $\{a_1, a_2, ..., a_i\}$ پیدا کرد که مجموعش j شود. $0 \leq j \leq s$. S(i,j) می‌تواند true یا false باشد. بر این اساس، می‌توان رابطه بازگشتی زیر را نوشت. دقت کنید OR همان یای منطقی است.

$$S(i,j) = \begin{cases} true & \text{if } i = j = 0 \\ false & \text{if i} = 0\ , j \neq 0 \\ S(i-1,j) & \text{if } j < a_i \\ S(i-1,j)\ \text{OR}\ S(i-1,j-a_i) & \text{otherwise} \end{cases}$$

شبه کد برنامه‌ریزی پویا برای رابطه‌ی بازگشتی به این صورت می‌باشد:

```
SubsetSum(A, s) {
    // Assume array indices are zero-based
    n = A.length
    S[0,0] = true
    for j = 1 to n {
        S[0,j] = false
    }
    for i = 1 to n {
        for j = 0 to s {
            S[i,j] = S[i-1,j]
            if j >= A[i-1]
                S[i,j] |= S[i-1, j-A[i-1]]
        }
    }
    return S[n, s]
}
```

پیچیدگی زمان اجرای الگوریتم، $\mathcal{O}(s \times n)$ است که شبه چندجمله‌ای است. پیچیدگی حافظه در این پیاده‌سازی، $\mathcal{O}(s \times n)$ است که قابل کاهش به $\mathcal{O}(s)$ می‌باشد.

---

۵. (۲۰ نمره) فرض کنید یک فایل با متن ABRACADABRA دارید (رنگ‌آمیزی حروف صرفا برای تسهیل در خواندن است).

الف) به کمک روش هافمن، محتوای این فایل را به صورت باینری کدگذاری کنید. مراحل انجام کدگذاری را بنویسید. نیازی به نوشتن شبه کد الگوریتم هافمن نیست.

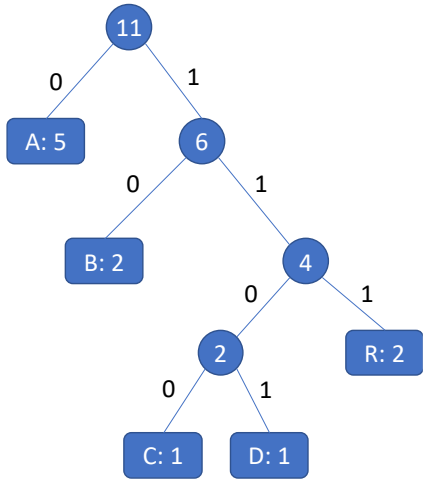
ب) میزان فشرده‌سازی (نسبت اندازه‌ی فایل فشرده شده به فایل اصلی) به کمک روش هافمن در مقایسه با ذخیره‌سازی به صورت متن چقدر است؟

ج) بهترین و بدترین میزان فشرده‌سازی برای هر رشته به کمک کدگذاری بدست آمده چقدر است؟ توضیح دهید.

نکته: هر بایت، ۸ بیت است.

پاسخ:

الف) (۱۰ نمره: ۶ نمره کد درست و درخت، ۴ نمره کدگذاری رشته) یکی از پاسخ‌های صحیح به کمک درخت زیر بدست می‌آید. پاسخ‌های درست دیگر با جابه‌جایی حروف با فراوانی مشابه (نظیر B و R) بدست می‌آیند.



حرف	A	B	C	D	R
کد	۰	۱۰	۱۱۰۰	۱۱۰۱	۱۱۱

بر این اساس، ABRACADABRA به صورت مقابل کد می‌شود:

۰۱۰۱۱۱۰۱۱۰۰۰۱۱۰۱۰۱۰۱۱۱۰

ب) (۴ نمره) دقت کنید که پاسخ این قسمت مستقل از جایگشت‌های جواب صحیح در قسمت (الف) است.

$$\frac{۲۳}{۱۱ \times ۸} = \frac{۲۳}{۸۸} \cong ۰.۲۶$$

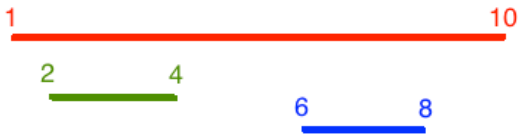
ج) (۶ نمره: ۳ نمره برای بهترین و ۳ نمره برای بدترین) بهترین فشرده سازی برای رشته‌هایی شامل حرف A اتفاق می‌افتد. در این رشته‌ها میزان فشرده‌سازی، $\frac{۱}{۸}$ است. بدترین فشرده‌سازی برای رشته‌هایی شامل C یا D اتفاق می‌افتد. میزان فشرده‌سازی این رشته‌ها، $\frac{۱}{۲}$ است.

---

۶. (۳۰ نمره) فرض کنید n بازه به صورت $I_i = [s_i, f_i]$ روی محور اعداد حقیقی داده شده‌اند. می‌خواهیم k نقطه به صورت $p_1, p_2, \ldots, p_k$ پیدا کنیم به طوری که به ازای هر i یک j وجود داشته باشد که $p_j \in I_i$. الگوریتم حریصانه‌ای پیشنهاد بدهید که مسئله را با کمترین k حل کند و شبه کد آن‌را بنویسید. بهینگی الگوریتم خود را ثابت کرده و پیچدگی زمان اجرای آن را بدست آورید.

پاسخ: ۱۰ نمره الگوریتم، ۷ نمره شبه کد، ۳ نمره تحلیل زمان اجرا، ۱۰ نمره اثبات

پاسخ حریصانه اشتباه: بازه‌ها را بر اساس زمان شروعشان مرتب می‌کنیم. سپس آن‌ها را به ترتیب بررسی می‌کنیم. اگر هر بازه با یک نقطه پوشانده نشده بود، نقطه شروع بازه را به لیست نقاط اضافه می‌کنیم. در شکل زیر، این راه ۳ نقطه ۱، ۲ و ۶ را انتخاب می‌کند در حالی که این مسئله با دو نقطه ۴ و ۸ قابل حل است.



پاسخ حریصانه صحیح: بازه‌ها را بر اساس زمان پایان‌یافتن مقایسه می‌کنیم. سپس به ازای هربازه اگر توسط نقطه‌ای پوشانده نشده بود، نقطه‌ی آخر بازه را به لیست نقاط اضافه می‌کنیم.

```
CoverIntervals(I) {
    I_sorted = sort_by_finish_times(I)
    points = []
    for each (s, f) in I_sorted {
        if points.size = 0 or points[-1] < s {
            points += [f]
        }
    }
    return points
}
```

زمان اجرای الگوریتم $\mathcal{O}(n \log n)$ می‌باشد.

اثبات بهینه بودن: فرض کنید راه حل حریصانه (S_{greedy}) به پاسخ بهینه نرسد. بدون کاسته شدن از کلیت مسئله، فرض می‌کنیم نقطه‌ها در هر راه حل به صورت صعودی مرتب شده‌اند. شبیه‌ترین راه بهینه به راه حل حریصانه (راهی که بیشترین تعداد نقطه مشابه از ابتدا را با راه حریصانه دارد) در نظر بگیرید و آن‌را S_{opt} بنامید. فرض کنید اولین نقطه‌ای که این دو راه حل با هم تفاوت دارند، در نقطه kام اتفاق می‌افتد:

S_{greedy}: $x_1, x_2, \ldots, x_k, \ldots$

S_{opt} : $y_1, y_2, \ldots, y_k, \ldots$

با توجه به نحوه طراحی الگوریتم حریصانه، می‌دانیم که $y_k < x_k$ زیرا در الگوریتم حریصانه آخرین نقطه ممکن برای پوشش بازه مورد نظر انتخاب می‌شود. اگر y_k را از S_{opt} حذف کرده و x_k را به آن اضافه کنیم، باز هم همان پوشش بازه‌های را خواهیم داشت. پس راه حل جدید بدست آمده هنوز بهینه است ولی به راه حریصانه شبیه‌تر شده است که این تناقض است. پس راه حل حریصانه، پاسخ بهینه را بدست می‌آورد.