طراحی الگوریتم (بهار ۱۴۰۱)

پاسخنامه امتحان اول

تاریخ امتحان: ۱۴۰۱/۱/۱۸

مدت امتحان: ۲ ساعت

توجه

- در مورد قوانین امتحان و نحوه‌ی ارسال پاسخ‌ها، به فایل «قوانین امتحان» مراجعه کنید.
- در سوال‌هایی که از شما طراحی الگوریتم خواسته شده است، تحلیل زمان اجرا و حافظه را بدست آورید.
- لطفا فقط از حاصل تلاش خود برای حل سوالات استفاده کنید و هیچ‌گونه کمکی به دانشجویان دیگر نکنید. در غیر اینصورت مشمول قوانین تقلب درس خواهید شد.
- در ابتدای اولین صفحه‌ی پاسخ‌ها، متن زیر را با خط خود نوشته و امضا نمایید:
«من تعهد می‌نمایم که پاسخ‌های داده شده حاصل تلاش شخصی من بوده و هیچ‌گونه کمکی دریافت نکرده یا به دیگران نرسانده‌ام.»

۱. (۲۰ نمره) فرض کنید یک آرایه به طول n از اعداد حقیقی به شما داده شده است. اگر بخواهید مقادیر کمینه و بیشینه این آرایه را پیدا کنید، ساده‌ترین راه انجام ۲(n-۱) مقایسه است: n-۱ مقایسه برای یافتن مقدار کمینه و n-۱ مقایسه برای یافتن مقدار بیشینه. هدف این سوال کاهش این تعداد مقایسه است.

الف) آیا می‌توانید تعداد مقایسه‌ها را کمتر از $O(n)$ نمایید؟ اگر بله، الگوریتم خود را توضیح دهید. در غیر این صورت، ثابت کنید چنین الگوریتمی وجود ندارد.

ب) در روش ذکر شده در صورت سوال، مسئله با $2n-2$ مقایسه قابل حل است. اگر این عبارت را به صورت $an+b$ بنویسیم، الگوریتمی پیشنهاد دهید که ضریب a را بهبود دهد (به عبارتی آن را کمتر از ۲ کند.)

پاسخ:

الف) (۵ نمره) مرتبه تعداد مقایسه‌ها قابل کاهش نیست زیرا همه‌ی اعضای آرایه باید به نحوی در مقایسه‌ها شرکت داده شوند در نتیجه هر الگوریتمی که ارائه شود از $O(n)$ خواهد بود.

ب) (۱۵ نمره: ۶ نمره الگوریتم، ۲ نمره تحلیل زمان و ۲ نمره تحلیل حافظه، ۵ نمره شبه کد) به کمک تقسیم و حل آرایه را می‌توان به دو قسمت تقسیم کرد. پس از بدست آوردن مقدار کمینه و بیشینه هر زیرآرایه، با انجام دو مقایسه (مقایسه دو مقدار بیشینه و کمینه) می‌توان مقدار کمینه و بیشینه‌ی کل را بدست آورد. تعداد مقایسه‌های انجام شده با رابطه بازگشتی زیر بدست می‌آید:

$$\begin{cases} T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T\left(\left\lceil\frac{n}{2}\right\rceil\right) + 2 \cong 2T\left(\frac{n}{2}\right) + 2 \\ T(2) = 1 \\ T(1) = 0 \end{cases}$$

برای حل این رابطه به صورت زیر عمل می‌کنیم:

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

اگر فرض کنیم $n=2^k$ خواهیم داشت:

$$T(n) = 2^{k-1}T(2) + 2^k - 2 = \frac{3}{2}n - 2$$

در این حالت $a = \frac{3}{2}$ است و برای حالاتی که n توانی از ۲ نباشد، مقدار بیشتر خواهد بود. میزان حافظه مصرفی، O(n) و حافظه کمکی O(۱) است.

```
FindMinMax(A) {
    n = A.size  // Assume n > 0
    if n == 1 {
        return A[0], A[0]
    } else {
        if A[0] < A[1]
            return A[0], A[1]
        else
            return A[1], A[0]
    }
    m0, M0 = FindMinMax(A[0, ..., n/2])
    m1, M1 = FindMinMax(A[n/2 + 1, ..., n - 1])

    m = m0 < m1 ? m0 : m1
    M = M0 < M1 ? M1 : M0

    return m, M
}
```

راه جایگزین: اگر n فرد باشد، عنصر اول را به عنوان کمینه و بیشینه در نظر بگیرید. اگر n زوج باشد، عنصر اول و دوم را با یک مقایسه به عنوان کمینه و بیشینه در نظر بگیرید. حال عناصر باقیمانده را به صورت زوج مقایسه کرده و با کمینه و بیشینه‌ی آنها را با کمینه و بیشینه بدست آمده مقایسه کنید (در مجموع ۳ مقایسه به ازای هر زوج.) با این روش در نهایت بیشینه و کمینه کل آرایه بدست می‌آید.

تحلیل پیچیدگی زمان اجرا:

n فرد باشد: 3(n-1)/2 = 3/2n - 3/2

n زوج باشد: 3(n-2)/2+2 = 3n/2 - 2

میزان حافظه مصرفی، O(n) و حافظه کمکی O(۱) است.

```
FindMinMax(A) {
    n = A.size  // Assume n > 0
    if n % 2 == 1 {
        m = A[0]
        M = A[0]
        start_index = 1
    } else {
        m = min(A[0], A[1])
        M = max(A[0], A[1])
        start_index = 2
    }
    for i in range(start_index, n, 2) {
        if A[i] < A[i + 1]
            temp_min = A[i]
            temp_max = A[i+1]
        else
            temp_min = A[i+1]
            temp_max = A[i]
        m = min(m, temp_min)
        M = max(M, temp_max)
    }
    return m, M
}
```

---

۲. (۳۰ نمره) فرض کنید k آرایه‌ی مرتب شده از اعداد حقیقی به شما داده شده است که طول هرکدام n است. از شما خواسته شده تا این آرایه‌ها را طوری ادغام کنید که آرایه‌ی نهایی مرتب و به طول $k \times n$ باشد.

الف) فرض کنید برای انجام ادغام از این روش استفاده می‌کنید: ابتدا آرایه‌ی اول و دوم را ادغام می‌کنید. سپس آرایه‌ی حاصل را با آرایه‌ی سوم ادغام کرده و نتیجه را با آرایه‌ی چهارم ادغام می‌کنید و همینطور تا آرایه‌ی nم ادامه می‌دهید. پیچیدگی زمان اجرای این روش بر حسب k و n چقدر است؟
ب) الگوریتمی با پیچیدگی زمانی کمتر برای ادغام آرایه‌ها ارائه داده و پیچیدگی زمانی آن را بیابید.

پاسخ:

الف) (۵ نمره) ادغام هر دو آرایه، نیاز به پیمایش هر دو آرایه دارد. در مرحله اول، دو آرایه به طول n با هم ادغام می‌شوند. سپس یک آرایه به طول $2n$ با یک آرایه به طول n، بعد یک آرایه به طول $3n$ با یک آرایه به طول n تا در نهایت یک آرایه به طول $(k-1)n$ با یک آرایه به طول n ادغام می‌شود. تعداد مقایسه‌ها در زیر محاسبه شده است.

$$(n+n)+(2n+n)+(3n+n)+\cdots+\big((k-1)n+n\big)$$
$$=\frac{k(k-1)}{2}n+(k-1)n=O(k^2n)$$

ب) (۲۵ نمره: ۱۶ نمره الگوریتم، ۲ نمره پیچیدگی زمانی، ۲ نمره پیچیدگی حافظه، ۵ نمره شبه کد) برای بهبود زمان اجرا، در هر مرحله، آرایه‌ها را دو به دو با هم ادغام می‌کنیم. سپس در مرحله‌ی بعد، آرایه‌های ادغام شده به طول $2n$ را دو به دو ادغام می‌کنیم. این کار ادامه پیدا می‌کند تا در نهایت دو آرایه‌ی به طول $kn/2$ ادغام کنیم. دقت کنید که در هر مرحله، همه‌ی عناصر در مقایسه شرکت می‌کنند پس پیچیدگی هر مرحله برابر kn است. تعداد مراحل نیز برابر $O(\log k)$ است. پس در نهایت با $O(kn \times \log k)$ مقایسه می‌توان ادغام را انجام داد. میزان حافظه مصرفی، $O(kn)$ است.

```
MergeArrays(A, k) {
    if k == 1 {
        return A
    }
    B = []
    for i = 0 to k / 2 - 1 {
        // Merge() is similar to what we have in MergeSort implementation
        B.append(Merge(A[2 * i], A[2 * i + 1]))
    }
    if k % 2 == 1 {
        B.append(A[k])
        MergeArrays(B, k / 2 + 1)
    } else {
        MergeArrays(B, k / 2)
    }
}
```

---

۳. (۲۵ نمره) برای انجام هر «زیرپروژه»، احتیاج به یک رئیس و یک مرئوس است و هیچ فردی نمی‌تواند در بیش از یک زیرپروژه مشارکت داشته باشد. شرکت «غول‌آسا» تصمیم دارد بیشترین تعداد زیرپروژه‌های ممکن را در مناقصه برنده شود. برای رسیدن به این منظور، تصمیم می‌گیرد بر اساس نمودار (چارت) سازمانی خود که به شکل یک درخت است، بیشترین زوج رئیس و مرئوس را انتخاب کرده و در مناقصه شرکت دهد. هر زوج رئیس و مرئوس بایستی در نمودار سازمانی غول‌آسا به طور مستقیم با هم در ارتباط باشند. به عبارتی، یک یال بین آن‌ها در درخت سلسله مراتب شرکت وجود داشته باشد. به شرکت غول‌آسا کمک کنید تا در زمان چندجمله‌ای (نسبت به تعداد رئوس درخت) تعداد بیشترین زوج رئیس و مرئوس را در درخت سلسله مراتب خود پیدا کند. درخت سلسله مراتب به صورت T_r (r ریشه‌ی درخت

یا همان رئیس شرکت) به شما داده شده است. نیازی به چاپ کردن رئوس شرکت‌کننده در جواب بهینه نمی‌باشد و صرفا تعداد زوج شرکت‌کننده را باید بدست آورید.

پاسخ: (۱۶ نمره توضیح الگوریتم، ۲ نمره پیچیدگی زمان، ۲ نمره پیچیدگی حافظه، ۵ نمره شبه کد)

مسئله را با برنامه‌ریزی پویا حل می‌کنیم. پاسخ برای هر زیردرختی با ریشه‌ی x را با OPT(x) نمایش می‌دهیم. در چنین زیردرختی، x می‌تواند بخشی از پاسخ باشد و یا نباشد. بر این اساس، ماکسیموم دو حالت زیر پاسخ مسئله خواهد بود:

الف) اگر x بخشی از پاسخ نباشد، حاصل برابر است با مجموع OPT(u) به طوری که u فرزند x است.
ب) اگر x بخشی از جواب باشد، ممکن است هرکدام از فرزندان x مثل u با آن بخشی از جواب باشند. برای همین حاصل برابر ماکسیموم جمع مقدار OPT(u) با مجموع OPT(v) برای رئوسی مثل v که فرزند x بوده ولی با u برابر نیستند می‌باشد.
این دو حالت به صورت ریاضی به شکل زیر نوشته می‌شوند:

$$OPT(x) = max\begin{cases} \Sigma_{u\in \text{children}(\text{x})} OPT(u) \\ 1 + max_{u\in \text{children}(\text{x})}\{\Sigma_{v\in \text{children}(\text{x}), \text{v}\neq \text{u}} OPT(v) + \Sigma_{v\in \text{children}(u)} OPT(v)\} \end{cases}$$

این رابطه بازگشتی را می‌توان به کمک برنامه‌ریزی پویا به صورت پایین به بالا حل کرد. حالت پایه برای برگ‌های درخت ۰ می‌باشد.

```
MaxParticipants(i) {
    OPT[i] = 0
    if i has no child
        return
    // i is not part of the solution
    opt1 = 0
    for j ∈ i’s children {
        MaxParticipants(j)  // DFS
        opt1 += OPT[j]
    }

    // i is not part of the solution
    opt2 = 0
    for j ∈ i’s children {
        // exclude j from the solution
        temp = opt1 - OPT[j]
        for k ∈ j’s children {
            temp += OPT[k]
        }
        opt2 = max(temp, opt2)
    }
    OPT[i] = max(opt1, opt2 + 1)
}
```

پیچیدگی زمان اجرای برنامه، $O(n^2)$ است که n تعداد رئوس درخت می‌باشد. پیچیدگی حافظه برنامه، $O(n)$ می‌باشد.
نکته: این مسئله راه‌حل حریصانه نیز دارد و در صورت اثبات درستی راه حریصانه، نمره‌ی کامل به شما تعلق می‌گیرد.

۴. (۳۵ نمره) دانشکده برق و کامپیوتر اخیرا یک کلاستر گران‌قیمت خریداری کرده است تا داده‌های آموزش مجازی را به صورت روزانه پردازش کند و مشکلات دانشگاه را یکبار برای همیشه در این زمینه برطرف کند. با توجه به تجربه دانشگاه در برگزاری کلاس‌های مجازی، می‌دانیم که حجم داده‌های دریافتی برابر $d_۱, d_۲, \dots, d_n$ به ترتیب برای روزهای ۱، ۲، ...، n می‌باشد. متاسفانه کار برنامه‌نویسی این کلاستر به دانشجویان کم تجربه‌ی سال اولی واگذار شده است و برنامه‌ای که آن‌ها برای پردازش داده نوشته‌اند طوری است که توان پردازش کلاستر هر روز افت می‌کند مگر آنکه کلاستر ریست (reset) شود. در روزی که کلاستر ریست می‌گردد، نمی‌توان داده‌ای را پردازش کرد. دانشجویان سال اولی، با سعی و خطا دریافته‌اند که اگر i روز از آخرین ریست کلاستر بگذرد، کلاستر توان پردازش p_i ترابایت داده را در آن روز خواهد داشت. آن‌ها جدولی تشکیل داده و مقادیر $p_۱ > p_۲ > \cdots > p_n > 0$ را در آن یادداشت کرده‌اند. رئیس دانشکده که از ضعف برنامه‌نویسی سال اولی‌ها به ستوه آمده از شما خواسته روزهایی که باید کلاستر را ریست کرد پیدا کنید تا طی n روز استفاده از کلاستر، بیشترین میزان داده پردازش شود. دقت کنید که:

أ) کلاستر پیش از تحویل به شما ریست شده است تا بالاترین کارایی را داشته باشد.

ب) همیشه میزان داده‌ی پردازش شده در یک روز برابر کمینه داده‌ی دریافتی و توان پردازش کلاستر در آن روز می‌باشد.

ج) داده‌های دریافتی هر روز باید فقط در همان روز پردازش شوند و شما نمی‌توانید آن‌ها را روز دیگری پردازش کنید.

د) با توجه به حساسیت موضوع، الگوریتم شما باید در زمان $O(n^2)$ اجرا شده و بیشینه داده‌هایی که پس از n روز می‌توان پردازش کرد به همراه لیستی از روزهایی که قرار است کلاستر ریست شود (به ترتیب از روز کم به زیاد) چاپ کند.

مثال: فرض کنید $n = 4$ و مقادیر d_i و p_i به صورت زیر به شما داده شده است:

	روز ۱	روز ۲	روز ۳	روز ۴
d	۱۰	۱	۷	۷
p	۸	۴	۲	۱

بهترین راه حل، ریست کردن کلاستر در روز ۲ است. در این حالت، می‌توانید ۸ ترابایت در روز ۱، ۰ ترابایت در روز ۲، ۷ ترابایت در روز ۳ و ۴ ترابایت در روز ۴ پردازش کنید که در مجموع ۱۹ ترابایت می‌باشد. اگر کلاستر را ریست نمی‌کردید، فقط ۸+۱+۲+۱ = ۱۲ ترابایت می‌توانستید پردازش کنید. ریست کردن کلاستر در روزهای دیگر نیز پاسخ بهتری از ۱۹ ترابایت به شما نمی‌دهد.

پاسخ: (۳۵ نمره: ۲۶ نمره الگوریتم،۲ نمره پیچیدگی زمان، ۲ نمره پیچیدگی حافظه، ۵ نمره شبه کد) این مسئله را با روش‌های مختلف برنامه‌ریزی پویا قابل حل است که در ادامه به دو روش اشاره می‌شود.

سوال یافتن روزهایی را خواسته که در آن‌ها ریست بایستی اتفاق بیفتد تا بیشترین پردازش داده در n روز انجام گیرد. در صورتی که بیشترین داده‌ی پردازش شده را هم بیابید، با ارفاق نمره‌ی کامل را می‌گیرید.

راه حل ۱: فرض کنید $OPT(i, j)$ برابر حداکثر میزان داده‌ای است که از روز iام تا روز nام می‌توان پردازش کرد به شرطی که آخرین ریست کردن j روز پیش از روز i (یا به عبارتی روز i-j) اتفاق افتاده باشد. در هر روز دو گزینه پیش رو داریم:

- ریست کردن: در این حالت هیچ داده‌ای در روز iام پردازش نمی‌شود و میزان داده‌ی پردازش شده برابر است با:

$$OPT(i, j) = OPT(i + 1, 1)$$

- ادامه پردازش: در این حالت، به میزان $min(d_i, p_j)$ داده پردازش می‌شود. به عبارتی داریم:

$$OPT(i, j) = min(d_i, p_j) + OPT(i + 1, j + 1)$$

تصمیم اینکه بهترین کار در هرروز کدام است از طریق محاسبه ماکسیمم دو عبارت بالا بدست می‌آید. دقت کنید که در روز آخر فایده‌ای ندارد که ریست انجام شود و بایستی داده‌ها پردازش شوند. در نتیجه داریم:

$$OPT(n,j) = min(d_n, p_j)$$

این حالت، پایه‌ی رابطه‌ی بازگشتی را تشکیل می‌دهد. پاسخ مسئله در $\mathrm{OPT}(1, 1)$ قرار دارد.
شبه کد مسئله در ادامه آمده است:

```
CalcMaxProcessedData(d, p) {
    for j = 1 to n {
        OPT[n, j] = min(d[n], p[j])
    }
    for i = n - 1 downto 1 {
        for j = 1 to i {
            OPT[i, j] = max(OPT[i+1, 1], min(d[i], p[j]) + OPT[i+1, j+1]))
        }
    }

    return OPT[1, 1]
}
```

به دلیل دو حلقه‌ی تو در تو و نیز انجام محاسبات زمان ثابت درون حلقه، پیچیدگی زمان اجرای مسئله، $\mathrm{O}(\mathrm{n}^2)$ است و میزان حافظه مصرفی $\mathrm{O}(\mathrm{n}^2)$ می‌باشد که قابل تقلیل به $\mathrm{O}(\mathrm{n})$ است (البته نیازی به اشاره به این مورد برای کسب نمره کامل نیست).

راه حل ۲:

فرض کنید $\mathrm{OPT}(\mathrm{i}, \mathrm{j})$ برابر حداکثر میزان داده‌ای است که از روز اول تا روز iام پردازش می‌شود به شرطی که آخرین ریست j روز پیش از روز فعلی اتفاق افتاده باشد.

وقتی $\mathrm{j} > 0$ است (به عبارتی آخرین ریست روز iام رخ نداده است)، به میزان $min(d_i, p_j)$ داده در روز پردازش می‌شود و داریم:

$$OPT(i,j) = OPT(i-1, j-1) + min(d_i, p_j)$$

وقتی $\mathrm{j} = 0$ است (به عبارتی ریست در روز iام اتفاق می‌افتد)، هیچ پردازشی در روز iام اتفاق نمی‌افتد. به علاوه ریست قبلی می‌تواند هر روزی پیش از روز iام اتفاق بیفتد. پس داریم:

$$OPT(i,0) = \max_{1 \le k \le i-1} \{OPT(i-1, k)\}$$

در حالت کلی می‌توان در عبارت بالا k را از 0 شروع کرد ولی فایده‌ای ندارد که دو روز پشت هم ریست انجام دهیم.

حالت پایه‌ی این رابطه‌ی بازگشتی برابر خواهد بود با $\mathrm{OPT}(0, \mathrm{j}) = 0$ به ازای هر مقدار $0 \le j \le n$.

به راحتی می‌توان الگوریتمی طراحی کرد که مقدار $\mathrm{OPT}(\mathrm{i}, \mathrm{j})$ را محاسبه کند و در نهایت مقدار $\max_{1 \le j \le n} \{OPT(n,j)\}$ را برگرداند. مشابه راه حل بالا، زمان اجرا برابر $\mathrm{O}(\mathrm{n}^2)$ و میزان حافظه مصرفی $\mathrm{O}(\mathrm{n}^2)$ خواهد بود که قابل تقلیل به $\mathrm{O}(\mathrm{n})$ است (البته نیازی به اشاره به این مورد برای کسب نمره کامل نیست).