

راهنمای گام به گام

به پروژه یک درس ساختمان داده‌ها و الگوریتم‌ها خوش آمدید!

در این راهنما قصد داریم قدم به قدم با هم پیش برویم و از ساده‌ترین روش، شروع به حل یک مسئله زیبا کنیم.

مطالعه‌ی راهنما قبل از کارگاه خالی از لطف نیست و سعی کرده‌ایم راهنما را طوری بنویسیم که به تنهایی و خارج از کارگاه هم بتوانید پیش بروید. در حین کارگاه هم از روی همین راهنما پیش خواهیم رفت. همچنین دستیاران آموزشی در کارگاه با شما خواهند بود تا در ابهامات و اشکالات احتمالی همراهیتان کنند.

مقدمه

همگی ما با مبانی برنامه‌نویسی C آشنایی داریم و احتمالاً به واسطه درس برنامه‌سازی پیشرفته با $Java$ هم آشنا شده‌ایم. همچنین هیچ بعید نیست عده‌ای از شما برنامه‌نویسی به زبان $Python$ را هم بلد باشید. در پروژه‌های این درس از زبان‌های C و یا $C++$ بهره خواهیم برد. (پایتون هم به پروژه صفر اضافه گردید.)

می‌دانستیم که هر برنامه برای اجرا مقدار مشخصی زمان نیاز دارد که این زمان تقریباً با تعداد دستورهای اجرا شده در آن برنامه متناسب است. در درس «ساختمان داده‌ها و الگوریتم‌ها» با مفهوم نماد O و احتمالاً Ω و Θ آشنا شدیم و حالا در این پروژه می‌خواهیم دست به کد شویم و زمان مورد نیاز برای اجرای راه‌حل‌های متفاوت با آردرهای زمانی متفاوت را به طور عملی ببینیم.

چاپ زمان اجرای کد

با یک سرچ ساده‌ی «زمان اجرا زبان فلان» یا «execution time folan language» می‌توان به سادگی مطالب زیر را پیدا کرد که ما برای راحتی شما و سرعت بخشیدن به کارگاه این کار را برایتان از قبل کرده‌ایم. با کلیک بر روی هر کدام از جعبه‌های زیر نحوه چاپ زمان اجرا را در زبان مورد نظر ببینید و حتماً در سیستم خودتان هم تست کنید! خروجی این برنامه‌ها به میلی‌ثانیه است.

▼ کد C++

```

1  #include <iostream>
2  #include <ctime>
3  using namespace std;
4
5  int main()
6  {
7      clock_t Start = clock();
8      //your code...
9      //int n;
10     //cin >> n;
11     //for(int i = 0; i < n; i++)
12     //    if(i%1000000 == 0)
13     //        cout << " ";
14     clock_t End = clock();
15     cout << int((double)(End - Start)/double(CLOCKS_PER_SEC))*1000 << "\n";
16 }

```

دقت کنید که ممکن است گاهی قسمت‌های زائدی از کد ما توسط کامپایلر حذف شود، یا بخش‌های ساده‌ای از کدمان توسط کامپایلر بهینه‌سازی شوند. مثلاً اگر یک حلقه خالی در این کد قرار بدهید مشاهده می‌کنید که هر چقدر شرط پایان حلقه بزرگ باشد باز هم کد در حدود 0 میلی‌ثانیه اجرا می‌شود.

▼ کد Python

```

1  import time
2  Start = int(round(time.time() * 1000))
3  #your code...
4  #b = int(0)
5  #for a in range(0, 5000000):
6  #    b = b+a
7  End = int(round(time.time() * 1000))
8  print(End-Start)

```

(با اجرای کد کامنت شده، و تغییر عدد ثابت آن می‌توانید حدود تعداد عملیاتی که سیستم شما در ۱ ثانیه اجرا می‌کند بدست آورید.)

صورت مسئله

دنباله‌ای از اعداد صحیح مثبت به طول n به ما داده شده است. به هر بازه‌ی پیوسته‌ی i تا j از دنباله یک «زیردنباله» می‌گوییم. پس n زیردنباله به طول ۱ داریم و همچنین $n - 1$ زیردنباله به طول ۲ داریم و ...

حال تمامی زیردنباله‌های ممکن را تصور کنید. خواسته‌ی مسئله، یافتن بازه‌هایی است که مجموع اعداد موجود در بازه حداکثر k باشد.

برای مثال به نمونه زیر دقت کنید:

```
index : [0  1  2  3  4]
numbers: 1  4  1  2  2
k : 4
```

در این مثال بازه‌هایی که جمعشان حداکثر ۴ است، به شرح زیر هستند: بازه‌ها با اندیس‌های:

$$[0] : sum = 1$$

$$[1] : sum = 4$$

$$[2] : sum = 1$$

$$[2, 3] : sum = 3$$

$$[3] : sum = 2$$

$$[3, 4] : sum = 4$$

$$[4] : sum = 2$$

در نتیجه جواب مسئله برابر ۷ است.

فرمت ورودی و خروجی مسئله

در خط اول ورودی دو عدد n و k با فاصله می‌آیند که به ترتیب نشان‌دهنده اندازه دنباله ورودی و حداکثر مجموع مورد نظر هستند. در خط بعدی n عدد صحیح با فاصله از یکدیگر می‌آیند. در خروجی کفایت تعداد بازه‌های با حداکثر مجموع k را چاپ کنید.

ورودی نمونه

```
5 4
1 4 1 2 2
```

خروجی نمونه

```
7
```

نحوه ارزیابی پروژه

همانطور که مشاهده می‌کنید پروژه بر اساس اینکه اندازه دنباله اولیه یعنی n تا چه اندازه بزرگ باشد به 3 زیرمسئله تقسیم شده است و همچنین در انتها یک نمودار نیز از شما خواسته شده است. ایده و راه‌حل هر ۳ زیرمسئله را در ادامه با هم می‌بینیم.

نمره‌ی زیرمسئله‌ها توسط داوری آنلاین کوئرا داده خواهد شد که در آن صحیح بودن خروجی کد شما در ازای تعدادی ورودی و مدت زمان اجرای کد شما برای هر ورودی سنجیده می‌شود.

نمره‌ی بخش «نمودار»، توسط دستیاران آموزشی داده خواهد شد. توضیحات بیشتر در مورد بخش «نمودار» در بخش خودش آورده شده است.

*توجه داشته باشید که در هر زیرمسئله راه حل با $O(n^2)$ (آردر) خواسته را پیاده‌سازی کنید چرا که راه‌حل‌های دیگر نمره‌ای نخواهد داشت. (به طور مثال در صورت بارگذاری کردن کد با استفاده از الگوریتم قسمت دوم در قسمت اول ، هر چند که نمره‌ی داوری آنلاین برای شما کامل باشد ، نمره‌ی دریافت نخواهید کرد. *

زیرمسئله یکم

اول از همه بیایید ساده‌ترین راه‌حل ممکن را برای مسئله پیاده‌سازی کنیم. دو متغیر i و j در نظر بگیرید به کمک این دو و با استفاده از حلقه‌ی تودرتو تمام شروع و پایان‌های ممکن را برای زیردنباله در نظر بگیرید. حال به ازای هر حالت از i و j ، تمام عناصر با اندیس‌های بین i تا j را با حلقه‌ای دیگر پیمایش کنید و مقادیر آن‌ها را با هم جمع کنید تا مجموع عناصر این زیردنباله بدست بیاید. حال به ازای هر زیر دنباله‌ای که مجموع اعضای آن حداکثر k باشد، مقدار 1 را به شمارنده‌ای دلخواه اضافه کنید. الگوریتم بالا را پیاده‌سازی و سپس تحلیل اُردر کنید.

▼ اُردر

الگوریتم فوق با نظر به اینکه ۳ حلقه‌ی تودرتو دارد از $O(n^3)$ می‌باشد.

محاسبه زمان تقریبی اجرا

محاسبه زمان تقریبی اجرای برنامه‌ها کار دشواری نیست. عموماً این موضوع به سیستمی که کد را اجرا می‌کند هم مربوط می‌شود اما یک استاندارد و حدود مشخصی دارد و در واقع می‌تواند نشان دهد که کدمان مثلاً یک ساعت زمان برای اجرا نیاز ندارد و در حدود یک ثانیه یا کمتر به جواب می‌رسد.

استاندارد حدودی این‌گونه است: تعداد عملیات‌های برنامه را می‌شماریم، در یک برنامه به زبان C یا C++ فرض می‌کنیم که هر $2 * 10^8$ عملیات در حدود یک ثانیه اجرا می‌شود. این عدد به سخت‌افزار و قدرت پردازش سیستم هم بستگی دارد. (توی پرانتز بگم که رزرو کردن حافظه در هنگام شروع اجرای برنامه هم تا حدی زمان نیاز دارد، مثلاً وقتی یک آرایه‌ی خیلی خیلی بزرگ تعریف می‌کنیم زمان اجرای برنامه هم زیاد میشه. فعلاً تا وقتی به مشکلش برخوردید این پرانتز رو نادیده بگیرید.)

در این سوال ما برنامه‌ای با حدود n^3 عملیات نوشتیم. به سوال «زیرمسئله یکم» بروید و محدودیت n و به خصوص حداکثر مقدارش را مشاهده کنید. آیا n^3 عملیات در کمتر از یک ثانیه انجام می‌شود؟

خب حالا با همین برنامه به سراغ «زیرمسئله دوم» بروید. سنگ مفت، گنجشک مفت، شاید اکسپت شد! البته به محدودیت n در این سوال هم گوشه چشمی داشته باشید. حدود n^3 عملیات با این n در یک ثانیه انجام می‌شود؟

زیرمسئله دوم

در این زیرمسئله اندازه ورودی بزرگتر خواهد بود و طبیعتاً الگوریتم قبلی ما جوابگوی حل آن در زمان مناسب نیست. در قسمت قبل برای به دست آوردن مجموع یک زیر بازه با شروع از i و پایان j تمامی اعداد موجود در این بازه را با هم جمع می‌کنیم که باعث طولانی شدن زمان اجرا می‌شود. برای جلوگیری از این کار می‌توانیم از مجموع زیر دنباله‌ی i تا $j - 1$ استفاده کنیم. بنابراین با نگهداشتن جمع i تا $j - 1$ و اضافه کردن a_j به آن، جمع زیر دنباله‌ی i تا j به دست می‌آید. بنابراین می‌توانیم یکی از حلقه‌های موجود را حذف کنیم و در نتیجه از زمان اجرای برنامه (به مقدار کافی) بکاهیم.

▼ آردر

الگوریتم جدید با نظر به اینکه ۲ حلقه‌ی تودرتو دارد از $O(n^2)$ می‌باشد.

زیرمسئله سوم!

شاید تعجب کنید اما این مسئله از این هم سریع‌تر می‌تواند حل شود (: اگر علاقه و وقت داشتید جا دارد تا چند ساعت روی این مسئله فکر کنید اگر هم هر یک را نداشتید، راه حل زیر را بخوانید و به اثبات انتهایی آن فکر کنید. در ادامه به حل می‌پردازیم.

برای ساده‌تر کردن بیان راه حل، فرض کنید به بازه‌هایی که در جواب باید شمرده شوند (جمع عناصرشان حداکثر k است) می‌گوییم بازه طلایی.

ابتدا فرض کنید سوال را به ازای همه بازه‌هایی مثل $[L, R]$ حل کرده‌ایم به شکلی که $R \leq i$. یعنی تعداد همه بازه‌های طلایی که $R \leq i$ دارند را شمرده‌ایم و می‌خواهیم باقی بازه‌ها را بشمریم.

در یک حرکت می‌خواهیم فرضمان را یک مرحله قوی‌تر کنیم. یعنی بازه‌های طلایی که $R = i$ دارند را بشمریم و به جواب قبلی اضافه کنیم. اگر مرحله به مرحله فرضمان را قوی‌تر کنیم و در نهایت به $i + 1$ برسیم، مسئله به طور کل حل می‌شود.

بازه‌های طلایی که $R = i$ دارند را در نظر بگیرید. این بازه‌ها چه ویژگی‌ای دارند؟

▼ لم اول

اگر بازه‌ای مانند $[L, R]$ طلایی باشد، بازه‌های

$$[L + 1, R], [L + 2, R], \dots, [R, R]$$

هم حتما طلایی هستند.

▼ اثبات

چون $a_j > 0$ پس داریم:

$$k \geq \sum_{j=L}^R a_j > \sum_{j=L+1}^R a_j > \sum_{j=L+2}^R a_j > \dots > \sum_{j=R}^R a_j$$

فرض کنید حداقل یک بازه طلایی با $R = i$ وجود دارد. طبق لم اول واضح است p_i وجود دارد به صورتی که همه بازه‌های

$$[p_i, i], [p_i + 1, i], \dots, [i, i]$$

طلایی هستند.

اگر هیچ بازه طلایی با $R = i$ وجود نداشته باشد، قرار دهید $p_i = i + 1$.

▼ لم دوم

$$p_{i-1} \leq p_i$$

علاوه بر فرض قبلی، فرض کنید حالا که تا $i - 1$ آمده‌ایم و مسئله را حل کرده‌ایم، فرض کنید مقدار $\sum_{j=p_{i-1}}^{i-1} a_j$ هم در متغیری مثل s نگه داشته‌ایم.

حال طبق لم دوم عمل کرده و سعی می‌کنیم p_i را پیدا کنیم. ابتدا حدس می‌زنیم $p_i = p_{i-1}$. برای اینکه حدس خود را آزمایش کنیم کافیست بررسی کنیم آیا $s + a_i \leq k$ ؟

اگر جواب بله باشد طبق لم دوم به نتیجه رسیده‌ایم و p_i به دست آمده‌است.

اگر جواب خیر باشد یعنی $p_i > p_{i-1}$ پس می‌توانیم با اطمینان، $a_{p_{i-1}}$ را از s حذف کنیم چون می‌دانیم این مقدار در بازه طلایی دیگری نخواهد آمد. با حدس دوباره‌ی $p_i = p_{i-1} + 1$ و آزمایش دوباره آن و پیش‌روی به همین ترتیب، در نهایت:

یا بازه‌ی مورد حدس‌مان بازه‌ای با $L > R$ خواهد بود که به وضوح در این حالت هیچ بازه طلایی با $R = i$ وجود ندارد.

یا به جوابی برای p_i میرسیم. پس $i - p_i + 1$ بازه طلایی جدید یافتیم. پس فرضمان را قوی کرده و به مرحله بعد ($i + 1$) پیش می‌رویم.

خب، حالا با این همه دردرس همچنان شاید بگویید که دو حلقه تو در تو (یکی برای i و یکی برای یافتن p_i) داریم پس لابد الگوریتم از $O(n^2)$ است. حرفتان درست است. الگوریتم از $O(n^2)$ است اما از $\Theta(n^2)$ نیست بلکه از $\Theta(n)$ است.

پیشنهاد می‌شود به اثبات این ادعا فکر کنید چون اثبات زیبایی دارد.

▼ اثبات

کافیست به دنباله‌ی p نگاه کنید. طبق لم دوم می‌دانیم این دنباله نازولی‌ست. برای یافتن p_i این دنباله از p_{i-1} استفاده کردیم و چندبار p_{i-1} را $1+$ کردیم تا بالاخره به نتیجه‌ای برای p_i برسیم. هر بار آزمایشمان هم از $O(1)$ زمان می‌گرفت. پس در مجموع می‌توان گفت برای به دست آوردن کل دنباله p ، به تعداد p_n بار هزینه از $O(1)$ داده‌ایم. پس در مجموع الگوریتم از $\Theta(p_n)$ است و می‌دانیم که $p_n \leq n + 1$. پس در نهایت الگوریتممان از $O(n)$ است.

زیرمسئله یکم

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

بر روی هر زیرمسئله فقط به راه‌حل‌های هم‌اُردر با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه‌حل زیرمسئله ۱ در زیرمسئله ۲ یا برعکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد.) این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئرا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد.

ورودی

ورودی شامل دو خط است. در خط اول دو عدد طبیعی n و k به ترتیب آمده است و در خط بعدی n عدد صحیح مثبت با فاصله از هم آمده‌اند که دنباله اعداد (آن را a می‌نامیم) است.

$$1 \leq n \leq 300$$

$$1 \leq k \leq 10^{18}$$

$$1 \leq a_i \leq 10^9$$

خروجی

خروجی برنامه‌ی شما باید تنها شامل یک عدد باشد که برابر تعداد بازه‌های با مجموع حداکثر k است.

مثال

ورودی نمونه ۱

5 4

1 4 1 2 2

خروجی نمونه ۱

7

ورودی نمونه ۲

9 8

5 3 10 2 4 5 7 6 2

خروجی نمونه ۲

11

زیرمسئله دوم

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

بر روی هر زیرمسئله فقط به راه‌حل‌های هم‌اُردر با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه‌حل زیرمسئله ۱ در زیرمسئله ۲ یا برعکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد.) این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئرا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد.

ورودی

ورودی شامل دو خط است. در خط اول دو عدد طبیعی n و k به ترتیب آمده است و در خط بعدی n عدد صحیح مثبت با فاصله از هم آمده‌اند که دنباله اعداد (آن را a می‌نامیم) است.

$$1 \leq n \leq 2000$$

$$1 \leq k \leq 10^{18}$$

$$1 \leq a_i \leq 10^9$$

خروجی

خروجی برنامه‌ی شما باید تنها شامل یک عدد باشد که برابر تعداد بازه‌های با مجموع حداکثر k است.

مثال

ورودی نمونه ۱

5 4

1 4 1 2 2

خروجی نمونه ۱

7

ورودی نمونه ۲

9 8

5 3 10 2 4 5 7 6 2

خروجی نمونه ۲

11

زیرمسئله سوم

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

بر روی هر زیرمسئله فقط به راه‌حل‌های هم‌اُردر با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه‌حل زیرمسئله ۱ در زیرمسئله ۲ یا برعکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد.) این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئرا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد.

ورودی

ورودی شامل دو خط است. در خط اول دو عدد طبیعی n و k به ترتیب آمده است و در خط بعدی n عدد صحیح مثبت با فاصله از هم آمده‌اند که دنباله اعداد (آن را a می‌نامیم) است.

$$1 \leq n \leq 2 \cdot 10^5$$

$$1 \leq k \leq 10^{18}$$

$$1 \leq a_i \leq 10^9$$

خروجی

خروجی برنامه‌ی شما باید تنها شامل یک عدد باشد که برابر تعداد بازه‌های با مجموع حداکثر k است.

مثال

ورودی نمونه ۱

5 4

1 4 1 2 2

خروجی نمونه ۱

7

ورودی نمونه ۲

9 8

5 3 10 2 4 5 7 6 2

خروجی نمونه ۲

11

نمودار

برای این بخش، کدهایی که در زیرمسئله‌های یکم تا سوم زدید را روی تست‌کیس‌هایی که در کانال قرار گرفته‌اند اجرا کرده و هر بار زمان اجرای کد را اندازه بگیرید. سپس برای هر زیرمسئله، منحنی زمان اجرای برنامه نسبت به n را رسم کنید. منحنی‌ها را در یک نمودار (که با هم قابل مقایسه باشند) و در قالب یک فایل زیپ یا pdf آپلود کنید.

(به طور پیش‌فرض فایل ارسالی شما محدودیت حجم ۵ مگابایتی دارد. اگر بیش این نیاز بود در گروه درس مطرح بفرمایید.)

نمودار (امتیازی)

این بار از شما می‌خواهیم روی هر کدام از راه‌حلهایتان تست‌ها (همان تست‌های سوال قبلی) را اجرا کنید و سه نمودار جدا بدست بیاورید که محور افقی آن n (اندازه دنباله ورودی) و محور عمودی آن «تعداد عملیات‌های کد» باشد.

برای شمارش تعداد عملیات‌های کد، فرض کنید کفایت تعداد عملیات‌های جمع، ضرب، تفریق و تقسیمی که برنامه شما انجام می‌دهد را بشمارید. برای مثال در کدهای زیر متغیر `cnt` این تعداد را می‌شمارد:

```

1  #include <stdio.h>
2  int cnt = 0;
3  int main(){
4      int a = 2*5 + 12 - 14;
5      cnt += 3;
6
7      int b = (a-3)/2;
8      cnt += 2;
9
10     int n;
11     scanf("%d", &n);
12     for(int i = 0; i < n; i++){//no need to count loop operations like i++
13         b = a-3;
14         cnt += 1
15     }
16     printf("%d\n", cnt);
17 }
```

```
cnt = 0
```

```
a = 2*5 + 12 - 14
```

```
cnt += 3
```

```
b = (a-3)/2
```

```
cnt += 2
```



```
8  
9  
10 n = input()  
11 for i in range(n):  
12     b = a-3  
13     cnt += 1  
14 print(cnt)
```

(به طور پیش فرض فایل ارسالی شما محدودیت حجم ۱۰ مگابایتی دارد. اگر بیش این نیاز بود در گروه درس مطرح بفرمایید.)