Step by step guide

> Welcome to the project of a lesson on building data and algorithms!

In this guide, we are going to go step by step together and start solving a beautiful problem in the simplest way.

Reading the guide before the workshop is not without merit, and we have tried to write the guide in such a way that you can proceed alone outside the workshop. We will follow this guide during the workshop. Also, the teaching assistants will be with you in the workshop to help you with any doubts and possible problems.

Introduction

All of us are familiar with the basics of *C* programming, and probably we are also familiar with *Java* through the course of advanced programming . Also, it is not unlikely that some of you know programming in *Python* . In the projects of this lesson , we will use *C* or *C++ languages* . (Python was also added to Safar project)

We knew that each program needs a certain amount of time to run, which is roughly proportional to the number of commands executed in that program. In the lesson "Data Structures and Algorithms" with the concept of symbols$O$And probablyΩAndΘWe got to know each other and now in this project we want to get to the code and practically see the time required to implement different solutions with different time orders.

Print code execution time

With a simple search "execution time Folan language" or "execution time Folan language" you can easily find the following content, which we have already done for you for your convenience and to speed up the workshop. By clicking on each of the boxes below, see how to print the

runtime in the desired language and be sure to test it on your own system! The output of these programs is in milliseconds.

▼ C++ code

```cpp
#include <iostream>
#include <ctime>
using namespace std;

int main()
{
  clock_t Start = clock();
  //your code...
  //int n;
  //cin >> n;
  //for(int i = 0; i < n; i++)
  //  if(i%1000000 == 0)
  //      cout << "";
  clock_t End = clock();
  cout << int((double(End - Start)/double(CLOCKS_PER_SEC))*1000) << "\n";
}
```

Please note that sometimes redundant parts of our code **may** be removed by the compiler, or simple parts of our code may be optimized by the compiler. For example, if you put an empty loop in this code, you will see that no matter how big the end condition of the loop is, the code will still be around0Runs in milliseconds.

▼ Python code

```python
import time
Start = int(round(time.time() * 1000))
#your code...
#b = int(0)
#for a in range(0, 5000000):
#    b = b+a

```

```
End = int(round(time.time() * 1000))
print(End-Start)
```

(By running the commented code, and changing its constant number, you can get about the number of operations that your system executes in 1 second.)

Problem

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 بازه‌ّهایی 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 کافیست 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(اُردر) 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 ، نمره‌ای 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. عموما 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 مثلا 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، مثلا وقتی یک آرایه‌ی خیلی خیلی بزرگ تعریف می‌کنیم زمان اجرای برنامه هم زیاد میشه. فعلا تا وقتی به مشکلش برنخوردید 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 طبیعتا الگوریتم قبلی ما جوابگوی حل آن در زمان مناسب نیست. در قسمت قبل برای به‌دست‌آوردن 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 زیردنباله‌ی i تا $j-1$ استفاده کنیم. بنابراین با نگه‌داشتن جمع i تا $j-1$ و اضافه‌کردن a_j به آن، جمع زیردنباله‌ی 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.

▼ اُردر

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، راه‌حل زیر را بخوانید و به اثبات انتهایی آن فکر کنید. در ادامه به حل می‌پردازیم.

برای ساده‌تر کردن بیان راه‌حل، 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

$$[L + 1, R], [L + 2, R], ..., [R, R]$$

هم حتما طلایی هستند.

▼ اثبات

چون $a_j > 0$ پس داریم:

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

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

$$[p_i, i], [p_i + 1, i], ..., [i, i]$$

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 $\sum_{j=p_{i-1}}^{i-1} a_i$ 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 دردسر 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 نانزولی‌ست. برای یافتن p_i این دنباله از p_{i-1} استفاده کردیم و چندبار p_{i-1} را +۱ 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 درنهایت الگوریتممان از $O(n)$ است.