باسمه تعالی

آزمون عملی نخست

ساختمان داده‌ها و الگوریتم‌ها

۲۲ فروردین ۱۳۹۸

پاسخ‌نامه

افتخارآفرینی سیدپارسا (دسته‌بندی: درخت‌ها)

محدودیت زمان: ۱ ثانیه

محدودیت حافظه: ۲۵۶ مگابایت

برای حل این سوال کافی است دقت کنید که اولین راس ورودی ریشه است؛ اگر این راس مثلا ۳ فرزند داشته باشد، آن‌گاه ۳ راس بعدی مربوط به راس‌های سطح ۲ درخت هستند؛ سپس اگر این سه راس به ترتیب ۱، ۳ و ۲ فرزند داشته باشند آن‌گاه ۶ راس بعدی مربوط به سطح ۳ درخت هستند؛ و به همین‌ترتیب اگر این رئوس مجموعا k فرزند داشته باشند آن‌گاه k راس بعدی مربوط به سطح بعدی هستند. با همین روند می‌توان تعداد سطح‌ها یا همان ارتفاع درخت را تعیین کرد. هزینه‌ی زمانی این الگوریتم از $\theta(n)$ است.

کد این برنامه به صورت زیر است:

```
n = int(input())
arr = [*map(int,input().split())]
lst_sum = 1
cur_sum = 0
pointer = 0
ans = 0

while pointer < n:
     cur_sum = 0
     for i in range(pointer,pointer+lst_sum):
          cur_sum += arr[i]
     pointer += lst_sum
     ans += 1
     lst_sum = cur_sum
print(ans)
```

دکتر شریفی vs طالبی (دسته‌بندی: ترای)

محدودیت زمان: ۰/۵ ثانیه
محدودیت حافظه: ۲۵۶ مگابایت

برای حل زیرمسئله‌ی اول کافی است به ازای هر رشته‌ای که اضافه می‌شود آن را با تک‌تک رشته‌های قبلی مقایسه کنیم تا ببینیم آیا پیشوند یکی از آن‌ها هست یا خیر.

برای حل زیرمسئله‌ی دوم و یا کل سوال باید رشته‌های ورودی را به ترتیب در یک ترای اضافه کنیم؛ اگر طول رشته‌ها به ترتیب نزولی باشد (زیرمسئله‌ی دوم) آن‌گاه به ازای هر رشته که اضافه می‌شود تنها حالتی که «تابلو» می‌شود این است که این رشته پیشوند یک رشته‌ای باشد که قبلا اضافه شده؛ پس بعد از اضافه‌شدن هر رشته، اگر راس فعلی ما در ترای (که راس منتناظر با آخرین حرف رشته است) دارای فرزند بود (یعنی رشته‌ای شامل رشته‌ی فعلی و بزرگ‌تر از آن قبلا اضافه شده) یا قبلا یک رشته به آن راس ختم شده‌بود (یعنی دقیقا این رشته قبلا وجود داشته) باید اعلام کنیم که وضعیت تابلو شده است! پس برای هر گره‌ی درخت یک متغیر بولین می‌گیریم که نشان دهد آیا رشته‌ای به این گره ختم شده است یا نه.

برای حل کل سوال باید این حالت اضافه را در نظر بگیریم که وقتی یک رشته اضافه می‌شود ممکن است یکی از رشته‌هایی که قبلا اضافه شده پیشوند آن بشوند؛ بنابراین همین متغیر بولین که در قسمت قبل گفته شد را در نظر بگیرید؛ اگر در تمام طول مسیر اضافه‌کردن کاراکترهای رشته‌ی جدید (نه فقط در گره‌ی متناظر به حرف آخر) راسی وجود داشت که این متغیر بولینش برابر با یک شد یعنی یک رشته قبلا وجود داشته که پیشوند رشته‌ی فعلی ما است. هزینه‌ی زمانی این الگوریتم از مرتبه‌ی مجموع طول رشته‌های ورودی است.

کد این برنامه به صورت زیر است:

```
MAX = 100000
mark = [False for i in range(MAX)]
nxt = [[-1 for i in range(MAX)] for i in range(26)]
cnt = 1

def add(msg):
	global cnt, nxt, mark
	ptr = 0
	for char in msg:
		ch = ord(char)-ord('a')
		if nxt[ch][ptr] == -1:
			nxt[ch][ptr] = cnt
			cnt += 1
		ptr = nxt[ch][ptr]
		if mark[ptr]:
			return True
	for i in range(26):
		if nxt[i][ptr] != -1:
			return True
	mark[ptr] = True
	return False

def end_program(msg):
	print(msg)
	exit()

txts = []
while True:
	txt = input()
	if len(txt) == 1 and txt[0] == "0":
		end_program("everything is alright")
	if add(txt):
		end_program(txt)
	txts.append(txt)
```

زیدان هم برگشت؛ اما تو نیامدی! (دسته‌بندی: استک)

محدودیت زمان: ۱ ثانیه

محدودیت حافظه: ۲۵۶ مگابایت

برای حل زیرمسئله در هر مرحله مینیمم اعداد را در نظر بگیرید، فرض کنید این مینیمم در اندیس i قرار داشته‌باشد؛ حال می‌توان a_iتا بازه $[1,n]$ انتخاب کرد و از تمامی اعداد a_iتا کم کرد. اکنون تعدادی از اعداد برابر صفر می‌شوند که آن ها را حذف می‌کنیم مسئله را در تکه‌آرایه‌های باقی‌مانده حل می‌کنیم. این راه‌حل از $O(n^2)$ است.

برای حل کل سوال فرض کنیم که بازه‌های جواب به صورت $[L_1, R_1]$، $[L_2, R_2]$ و ... $[L_m, R_m]$ هستند. حال دو عنصر کنار هم a_i و a_{i+1} را در نظر بگیرید؛ اگر $a_{i+1} > a_i$ یعنی i در a_iتا از بازه‌ها و $i+1$ در a_{i+1}تا از آن‌ها حضور دارند؛ در نتیجه باید $a_{i+1} - a_i$ بازه از اندیس $i+1$ **شروع شوند**. زیرا اگر فرض کنیم یک بازه‌ای در اندیس i بسته شود (مثلاً بازه‌ی $[x, i]$) و بازه‌ای دیگر در $i+1$ باز شود (مثلاً $[i+1, y]$) می‌توان این دو بازه را ادغام کرد (به صورت $[x, y]$) و بدون تغییر در آرایه‌ی a تعداد بازه‌های نهایی کمتر شوند. پس به‌صرفه است صرفاً به اندازه اختلاف این دو عنصر مجاور بازه از اندیس $i+1$ شروع کنیم. به صورت مشابه اگر $a_{i+1} \leq a_i$ تعداد $a_i - a_{i+1}$ بازه در i **بسته می‌شوند**.

پس ما مکان بسته شدن و باز شدن بازه‌ها را داریم. اگر شرط داخل هم بودن بازه‌ها و یا مجزا بودنشان را در نظر نگیریم، می‌توان به هر روش دل‌خواه یکی از مکان‌های بازشدن را با یکی از مکان‌های بسته‌شدن متناظر کرد و یک بازه (برای خروجی) ساخت. اما با توجه به شرایط گفته‌شده در سوال می‌توانیم مسئله را شبیه سوال پرانتزگذاری در نظر بگیریم؛ یعنی به ازای هر باز شدن بازه یک کاراکتر) و هر بسته شدن یک کاراکتر (قرار دهیم، اکنون سوال تبدیل به این می‌شود که به هر پرانتز باز ، یک پرانتز بسته نسبت دهیم به صورتی که در نهایت تبدیل به یک پرانتزگذاری صحیح شود که این سوال با استفاده از استک قابل حل است.

کد سوال به صورت زیر است:

```
n = int(input())
arr = [*map(int,input().split())]
arr = [0] + arr + [0]
st = []
ans = []
for i in range(n+1):
     x = arr[i]
     y = arr[i+1]
     while x > y:
          x -= 1
          ans.append((st.pop(),i))
     while x < y:
          x += 1
          st.append(i+1)
print(len(ans))
for pair in ans:
     print(pair[0], pair[1])
```