دانشکده‌ی علوم ریاضی

داده‌ساختارها و الگوریتم‌ها ۷ دی ۱۳۹۶

گزارش پروژه اول

مدرّس: دکتر شهرام خزائی نگارنده: علی عمادی - ۹۵۱۰۰۱۹۱

۱ مساله ۱

ابتدا با استقرا ثابت می‌کنیم که حتماً چنین شهری وجود دارد.
پایه: $n = 1$
از آنجایی که $\sum c_i = \sum l_i$ بنابراین با شروع از تنها شهر موجود می‌توان یک دور،دور زد.
گام:
فرض می‌کنیم برای n شهر،شهر مورد نظر مساله وجود داشته باشد.حال ثابت می‌کنیم چنین شهری برای $n + 1$ شهر هم وجود دارد.
در بین $n + 1$ شهر حتماً شهری وجود دارد که از آن بتوان به شهر بعدی رفت،زیرا اگر چنین شهری وجود نداشته باشد آنگاه با فرض $\sum c_i = \sum l_i$ مساله در تناقض است.فرض کنید این شهر،iامین شهر باشد.آنگاه بنزین شهر $i + 1$ام را در شهر iام می‌ریزیم و شهر $i + 1$ام را حذف می‌کنیم.حال n شهر داریم که طبق فرض استقرا شهر مناسب مسئله وجود دارد.بنابراین چنین شهری همیشه وجود دارد.
در الگوریتم استفاده شده در کد،ما زیرآرایه بیشینه را برای ورودی پیدا می‌کنیم.خانه شروع زیرآرایه بیشینه جواب مسئله ماست.تنها تفاوت این مسئله با مسئله زیرآرایه بیشینه این است که در اینجا چون مسیر دایره‌ای است،زیرآرایه هایی مانند $M[n-k]...M[n]A[1]...M[r]$ را هم باید به حساب بیاوریم.ازاین پس زیرآرایه هایی که از خانه شروع و آخر نمی‌گذرند را زیرآرایه خطی و زیرآرایه هایی را که از این دو خانه می‌گذرند را زیرآرایه دوری (!) می‌نامیم.
ابتدا آرایه جدیدی مانند $M[1...n]$ تعریف می کنیم به طوری که:

$$M[i] = c[i] - l[i]$$

به ازای $1 \leq i \leq n$.به این ترتیب عضو iام M میزان تغییر بنزین ماشین بعد از طی مسیر بین شهر iام و $i + 1$ام را نشان می‌دهد.واضح است که چون $\sum c_i = \sum l_i$ داریم : $\sum M[i] = 0$.
بنابراین لازم است یک بار زیرآرایه بیشینه خطی را در $O(n)$ (که شبه‌کد آن آمده) و یک بار هم زیر آرایه بیشینه دوری را حساب کنیم،البته دقت کنید زیرآرایه بیشینه دوری حتماً شامل زیرآرایه بیشینه خطی هم هست زیرا اگر نباشد با توجه به اینکه $\sum M[i] = 0$ با اضافه کردن زیرآرایه بیشینه خطی به آن می‌توانیم به مجموع بیشتری برسیم.
شبه کد:

```
function Q1(L[1...n], C[1...n])
    Let M[1...n] be new array
    Let A[1...n] be new array
    Let B[1...n] be new array
    for i = 1 to n do
        M[i] = C[i] − L[i]
    A[begin...end] = findLinearMS(M[1...n]) //find maximum subarray, begin = first index of maximum subarray in M
    B[m...r] = findCircularMS(M[1...n]) //find maximum circular subarray, m = first index of maximum circular subarray in M
    for i = begin to end do
        l = l + A[i]
    for i = m to r do
        c = c + B[i]
    if l > c then
        return begin
    else
        return m
```

شبه کد برای پید کردن زیرآرایه بیشینه خطی با زمان خطی :

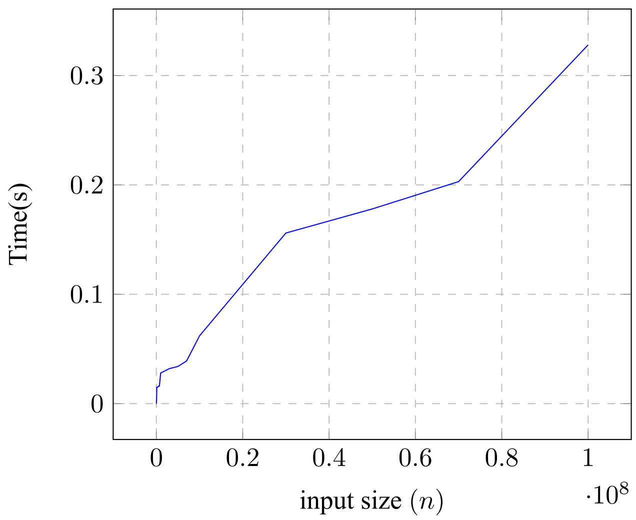
```
function MaximumSubarray(A[1...n])
    maxSoFar = −INF
    maxEndingHere = 0
    for i = 1 to n do
        maxEndingHere = maxEndingHere + A[i]
        if maxSoFar < max_ending_here then
            maxSoFar = maxEndingHere
        if maxEndingHere < 0 then
            maxEndingHere = 0
    return maxSoFar
```

این تابع هر بار زیرآرایه بیشینه را برای زیرآرایه هایی که به $A[i]$ ختم می‌شوند محاسبه می‌کند و بزرگترین آن ها را در $maxSoFar$ قرار می‌دهد.

کارآرایی کد:

جدول ۱: زمان برحسب طول ورودی

n	time
1000	0
10000	0
50000	0.003
100000	0.015
300000	0.015
500000	0.016
700000	0.016
1000000	0.028
3000000	0.032
5000000	0.034
7000000	0.039
10000000	0.062
30000000	0.156
50000000	0.178
70000000	0.203
100000000	0.328



۲ مساله ۲

فرض کنید ورودی به شکل آرایه $A[]$ باشد.ما به دنبال سه‌تایی هایی هستیم به طوری که:

$$A[i] > A[j] > A[k], i < j < k$$

یعنی وارونگی های به طول سه.برای پیدا کردن چنین وارونگی‌هایی فرض می کنیم به هر عضو مانند عضو jام آرایه که می رسیم این عضو،عضو وسط این وارونگی باشد.بنابراین برای هر عضو آرایه کافی است تعداد اعداد بزرگ‌تر سمت چپ و کوچک‌تر سمت راست را بشماریم و سپس این دو عدد را در هم ضرب کنیم تا تعداد وارونگی هایی که j عضو وسط آن است به دست آید.
برای مثال آرایه زیر در نظر بگیرید:

$$A = [۳, ۵, ۲, ۴, ۱]$$

فرض کنید در حال بررسی عضو سوم آرایه یعنی ۲ هستیم،ابتدا تعداد اعداد کوچک‌تر سمت راست آن را میشماریم که برابر است با ۱ و سپس تعداد اعداد بزرگ‌تر سمت چپ ۲ را میشماریم که این اعداد ۲ تا هستند، سپس ۲ را در ۱ ضرب می‌کنیم.به این ترتیب تعداد وارونگی های به طول ۳ که عضو وسطشان ۲ باشد،۲ تا هستند.(یعنی $[۳, ۲, ۱]$ و $[۵, ۲, ۱]$).

برای شمردن تعداد اعداد کوچک تر سمت راست یک آرایه از $MergeSort$ استفاده می‌کنیم.سپس آرایه را برعکس می کنیم و تعداد اعداد کوچک تر سمت راست همان عنصر را در آرایه برعکس میشماریم.سپس این عدد را منهای تعداد کل اعداد سمت راست این آرایه می کنیم تا تعداد اعداد بزرگ‌تر سمت راست در آرایه برعکس شده به دست آید.که این عدد دقیقاً همان تعداد اعداد بزرگ‌تر سمت چپ آن عنصر در آرایه اصلی است. حال که این دو عدد را داریم فقط کافی است که آن دورا در هم ضرب کنیم تا تعداد وارونگی های به طول ۳ که این عنصر از آرایه عضو وسط آن باشد را داشته باشیم.

چگونگی شمردن تعداد اعداد کوچک‌تر سمت راست با استفاده از $MergeSort$:

ایده اصلی این است که به جای مرتب کردن خود آرایه اصلی ، اندیس های مربوط به آن ها را مرتب کنیم.به عنوان مثال آرایه $[۵, ۲, ۳, ۸]$ را در نظر بگیرید.آرایه اندیس های اولیه مانند $[۱, ۲, ۳, ۴]$ خواهد بود و پس از مرتب شدن آرایه اصلی،این آرایه به صورت $[۳, ۱, ۲, ۴]$ درخواهد آمد.

هنگام انجام قسمت $merge$ دو زیرآرایه $left[]$ و $right[]$ (که خود مرتب شده‌اند)،تنها کاری که باید انجام دهیم این است که هربار که عنصری از $left$ به آرایه مرتب شده جدید انتقال پیدا می‌کند،تعداد اعدادی که تا این لحظه از $right$ درون آرایه مرتب شده رفته‌اند را بشماریم.(که چون زودتر از این عضو $left$ در آرایه قرار گرفته‌اند پس از آن کوچک تر هستند.)

شبه کد:

---

function COUNTSMALLER($A[0...n-1]$)
  Let $count[0...n-1]$ and $indexes[0...n-1]$ be new arrays
  for $i = 0$ **to** $n-1$ **do**
    $indexes[i] = i$
  $mergeSort(A, indexes, 0, n-1, count)$
  return $count$

---

function MERGESORT($A[0...n-1], indexes[0...n-1], start, end, count[0...n-1]$)
  if $end \leq start$ **then**
    return
  $mid = \frac{start+end}{2}$
  $mergeSort(A, indexes, start, end, count)$
  $mergeSort(A, indexes, mid+1, end, count)$
  $merge(A, indexes, start, end, count)$

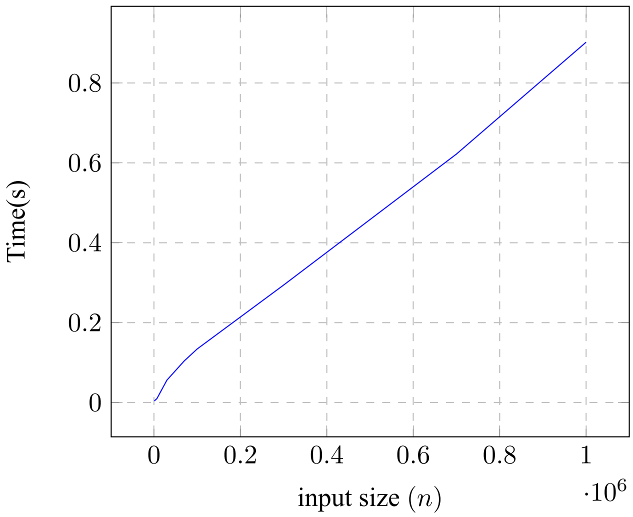
```
function MERGE(A[0...n−1], indexes[0...n−1], start, end, count[0...n−1])
    mid = (start+end)/2
    left = start
    right = mid + 1
    rightCount = 0
    sortIndex = 0
    newIndexes new array of length (end − start + 1)
    while left ≤ mid and right ≤ end do
        if A[indexes[right]] ≤ A[indexes[left]] then
            newIndexes[sortIndex] = indexes[right]
            rightCount = rightCount + 1
            right = right + 1
        else
            newIndexes[sortIndex] = indexes[left]
            count[indexes[left]] = count[indexes[left]] + rightCount
            left = left + 1
        sortIndex = sortIndex + 1
    while left ≤ mid do
        newIndexes[sortIndex] = indexes[left]
        count[indexes[left]] = count[indexes[left]] + rightCount
        left = left + 1
        sortIndex = sortIndex + 1
    while right ≤ end do
        newIndexes[sortIndex] = indexes[right]
        right = right + 1
        sortIndex = sortIndex + 1
    for i = start to end do
        indexes[i] = newIndexes[i − start]
```

چون از $MergeSort$ استفاده می‌کنیم بنابراین این الگوریتم از $\theta(n \lg n)$ است.

کارآرایی کد:

جدول ۲: زمان برحسب طول ورودی

n	time
1000	0.004
3000	0.006
7000	0.01
10000	0.016
30000	0.056
70000	0.104
100000	0.134
300000	0.294
700000	0.622
1000000	0.902



۳ مساله ۳

در این سوال از یک صف دوطرفه استفاده شده است.صف دوطرفه مانند همان صف یک‌طرفه است با این تفاوت که می‌توان اعداد را به اول یا آخر صف اضافه کرد و یا از اول یا آخر حذف کرد و یا عضو اول یا آخر را خواند.
ایده اصلی این است که ما فقط عناصری را نگه می‌داریم که در یک kتایی مشخص از تمام عناصر سمت چپشان بزرگتر باشند.
هنگام اضافه کردن ورودی iام به صف دوطرفه،ابتدا چک می‌کنیم که آیا این ورودی از عضو آخر صف بزرگتر است یا خیر.اگر بزرگتر بود عضو آخر صف را حذف می‌کنیم و این کار را آنقدر ادامه می‌دهیم تا یا صف خالی شود و یا عضو آخر از ورودی iام بزرگتر باشد.آنگاه ورودی را به آخر صف اضافه می‌کنیم.
به این ترتیب ما در هر مرحله در صفمان فقط اعدادی مانند اعداد ذکر شده در بند دوم را نگه می‌داریم.نکته مهم‌تر این است که این اعداد به ترتیب نزولی از جلوی صف به آخر صف مرتب می‌شوند.یعنی همیشه بزرگترین مقدار موجود در صف در جلوی صف قرار دارد و کوچکترین عدد موجود در آخر صف قرار دارد.نکته دیگر این است که هربار باید چک کنیم که آیا عددی از پنجره kتایی خارج شده است یا نه و اگر خارج شده بود آن را از صف حذف کنیم.
شبه کد:
شبه کد در اینجا با استفاده از آرایه نوشته شده اما در کد از آرایه استفاده نشده است.دقت کنید برای اینکه چک کردن اینکه آیا عنصری از پنجره خارج شده یا نه راحت‌تر باشد،به جای اضافه کردن خود عدد به صف،اندیس آن را اضافه می‌کنیم.

```
function Q3(A[0...n − 1], k)
    let Q bew new Dequeue
    for i = 0 to k do
        while !isEmpty(Q) and A[peekLast(Q)] ≤ A[i] do
            removeLast(Q)
        addLast(Q, i)
    for i = k + 1 to n − 1 do
        print(A[peekFirst(Q)])
        while !isEmpty(Q) and peekFirst(Q) ≤ i − k do
            removeFirst(Q)
        while !isEmpty(Q) and A[peekLast(Q)] ≤ A[i] do
            removeLast(Q)
        addLast(Q, i)
    print(A[peekFirst(Q)])
```

.

کارآرایی کد:

جدول ۳: زمان برحسب طول ورودی

n	time
1000	0.001
3000	0.002
7000	0.004
10000	0.005
30000	0.012
70000	0.021
100000	0.029
300000	0.035
700000	0.054
1000000	0.067
3000000	0.152
7000000	0.222
10000000	0.361
30000000	1.023
70000000	1.868