پاسخ تمرین شماره ۵

دانشکده مهندسی برق و کامپیوتر

ساختمان داده - بهار ۱۳۹۹

استاد : **فتحیه فقیه**

مسئول تمرین : **امین اسدی**
aminasadi329@gmail.com

۱.

الف) با استفاده از کاوش مرتبه۲ جدول hash در هر مرحله به صورت زیر خواهد بود:

Insert 10:											10
Insert 22:	22										10
Insert 31:	22									31	10
Insert 4:	22				4					31	10
Insert 15:	22				4				15	31	10
Insert 28:	22				4		28		15	31	10
Insert 17:	22			17	4		28		15	31	10
Insert 88:	22		88	17	4		28		15	31	10
Insert 59:	22		88	17	4		28	59	15	31	10

ب) با استفاده از درهم بندی دوگانه جدول hash در هر مرحله به صورت زیر خواهد بود:

Insert 10:											10
Insert 22:	22										10
Insert 31:	22									31	10
Insert 4:	22				4					31	10
Insert 15:	22				4	15				31	10
Insert 28:	22				4	15	28			31	10
Insert 17:	22			17	4	15	28			31	10
Insert 88:	22			17	4	15	28	88		31	10
Insert 59:	22		59	17	4	15	28	88		31	10

۲.الف)

	0	1	2	3	4	5	6	7	8	9	10
A =	6	0	2	0	1	3	4	6	1	3	2

ابتدا با هزینه‌ی خطی تعداد تکرار هر کدام از اعداد آرایه را در آرایه ای به نام C ذخیره می‌کنیم:

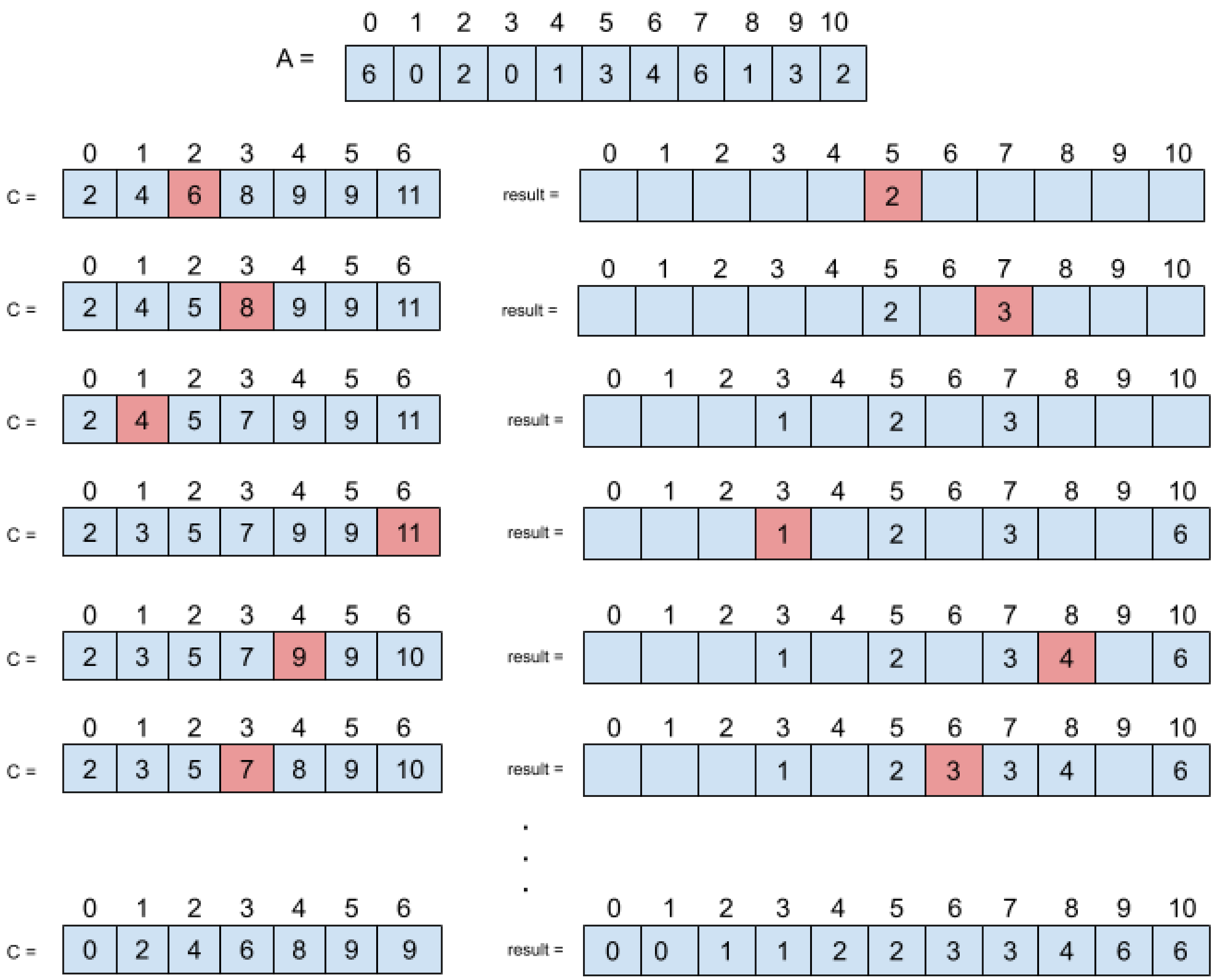
	0	1	2	3	4	5	6
C =	2	2	2	2	1	0	2

حال با هزینه‌ی خطی تعداد تکرارها را به صورت تجمعی ذخیره میکنیم:

	0	1	2	3	4	5	6
C =	2	4	6	8	9	9	11

حال با شروع از انتهای آرایه‌ی A آرایه‌ی result را به صورت زیر پر می‌کنیم:

```
for i = (n to 1):
   result[C[A[i]]-1] = A[i]
   C[A[i]] --
```

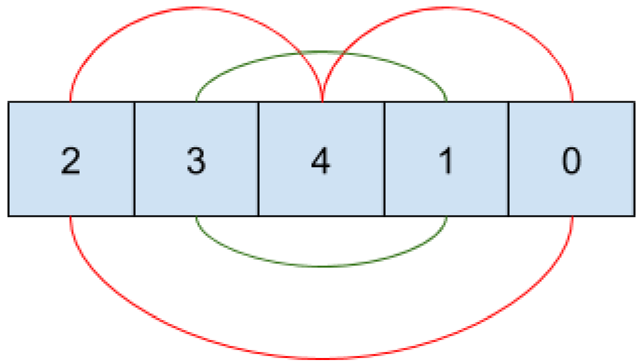


ب) بله این روش یک الگوریتم مرتب سازی پایدار است.

۳.

ابتدا آرایه‌ی B را به این صورت تغییر می‌دهیم که به جای هر عدد در آرایه‌ی B، ایندکس همان عدد در آرایه‌ی A را در B ذخیره می‌کنیم. برای مثال اگر دو آرایه $B = [3, 2, 4, 5, 1]$ و $A = [1, 5, 3, 2, 4]$ باشند، آنگاه آرایه‌ی B به آرایه‌ی $B' = [2, 3, 4, 1, 0]$ تبدیل خواهد شد. حال واضح است که در صورتی که آرایه‌ی B' را مرتب شود به جای هر عضو i ام آن مقدار $A[B[i]]$ جاگذاری شود، دو آرایه با هم یکسان خواهند شد. حال مسأله این است آرایه‌ی B' با کمترین تعداد جابجایی های ممکن

مرتب شود. در صورتی که جابجایی‌های لازم را رسم کنیم واضح است که جابجایی‌ها به صورت حلقه‌های بسته خواهند بود. برای مثال در مثال ذکر شده در بالا، جابجایی‌های لازم طبق شکل زیر با دو حلقه‌ی قرمز به طول ۳ و حلقه‌ی سبز به طول ۲ خواهند بود:



به وضوح در هر حلقه به طول x تعداد جابجایی‌های لازم برابر x-1 می‌باشد.پس تعداد جابجایی های لازم برابر خواهد بود با:

$$(3-1)+(2-1)=3$$

بنابراین برای یافتن حواب نهایی یعنی ans که در ابتدا مقدار آن برابر 0 است، کافیست طول حلقه‌های لازم را یافته و از آن‌ها یکی کم کنیم و با ans حمع کنیم. برای یافتن طول حلقه‌ها به این صورت عمل میکنیم که یک آرایه visited در نظر می‌گیریم که در ابتدا همه‌ی خانه‌های آن 0 هستند.آرایه‌ی B' را با شروع از ابتدای آن پیمایش می‌کنیم. وقتی به خانه‌ی i ام رسیدیم اگر قبلا آن خانه را بررسی کرده بودیم(یعنی خانه‌ی i ام آرایه‌ی visited برابر ۱ بود) یا اینکه عدد i ام آرایه B' دقیقا سر جای خود قرار داشت پیمایش را ادامه می‌دهیم. در غیر این صورت به صورت کد زیر عمل میکنیم و اندازه‌ی حلقه یعنی cycle_size را بدست می‌آوریم:

```
cycle_size = 0
  j = i
  while not vis[j]:
      # mark node as visited
      vis[j] = True

      # move to next node
      j = B'[j]
      cycle_size += 1
  ans += (cycle_size - 1)
```

که در اینجا در خط آخر ans با (cycle_size - 1) که در واقع برابر تعداد جابجایی‌های لازم در آن حلقه است جمع شده است.

پیمایش آرایه‌ی B' را به همین ترتیب تا آخر ادامه می‌دهیم و اندازه‌ی همه‌ی حلقه‌ها را بدست آورده از آن‌ها یکی کم کنیم و با ans حمع کنیم. در نهایت ans برابر جواب مسئله خواهد بود.

.۴

فرض کنید هر کدام از اعداد ورودی d رقم داشته باشند. با استفاده از radix-sort با پیچیدگی زمانی $O(d*(n+b))$ می‌توان اعداد را مرتب کرد که b در اینجا مبنای اعداد است. چون اعداد حداکثر n^2-1 هستند، d برابر خواهد بود با: $O(logb(n))$ (لگاریتم n در مبنای b) و در نتیجه پیچیدگی زمانی برابر خواهد بود با: $O((n+b)*O(logb(n))$. برای اینکه این پیچیدگی را خطی کنیم کافیست b را برابر n بگیریم تا $O(logb(n)) = O(1)$ شود و در نتیجه پیچیدگی زمانی برابر خواهد بود با: $O(n)$. دقت کنید که با توجه به اینکه اعداد حداکثر n^3-1 هستند پس در مبنای n حداکثر ۳ رقمی خواهند بود پس ۳ بار تابع count-sort صدا زده خواهد شد(برای هر رقم یک بار)

دقت شود که به جای تبدیل مبنای اعداد به مبنای n به این صورت عمل می‌کنیم که:

برای بدست آوردن رقم صفرم عدد x در مبنای n، کافیست x را بر 1 تقسیم کنیم و باقیمانده ی حاصل را بر n بدست آوریم.

برای بدست آوردن رقم اول عدد x در مبنای n، کافیست x را بر n تقسیم کنیم و باقیمانده ی حاصل را بر n بدست آوریم.

برای بدست آوردن رقم دوم عدد x در مبنای n، کافیست x را بر n^2 تقسیم کنیم و باقیمانده ی حاصل را بر n بدست آوریم.

.۵

اگر به ازای هر کدام از زوج مرتب ها مانند (u,v) یک بار DFS بزنیم آنگاه مشخص می‌شود که آیا مسیری از ریشه تا برگ درخت داده شده وجود دارد به طوری که u و v روی آن مسیر باشند یا خیر ولی در این صورت پیچیدگی زمانی از مرتبه $O((V+E)k)$ خواهد بود. راه حل این است فقط یکبار درخت را با DFS پیمایش کنیم و به ازای هر گره زمان ورود به گره (**Intime**) و زمان خروج از گره (**Outtime**) را بدست آورده و ذخیره کنیم. حال با کمی دقت مشخص می‌شود برای اینکه u و v روی یک مسیر باشند باید یک از دو شرط زیر برقرار باشد:

```
1) Intime[v] < Intime[u] and Outtime[v] > Outtime[u]
```

که در این صورت در مسیر، راس v به ریشه نزدیک تر است.

```
2) Intime[u] < Intime[v] and Outtime[u ]> Outtime[v]
```

که در این صورت در مسیر، راس u به ریشه نزدیک تر است.

بنابراین کافیست با هزینه $o(V+E)$ با DFS درخت را پیمایش می‌کنیم و زمان‌های ورود و خروج را بدست می‌آوریم و سپس با هزینه‌ی $o(k)$ روی آرایه‌ی ورودی پیمایش میکنیم و به ازای هر زوج مرتب مانند (u,v) شرط بالا را بررسی می‌کنیم. پس هزینه‌ی کل برابر است با:

$O(V+E+k)$

۶.

ابتدا یک بار بار شروع از یک گره دلخواه (این گره را گره شروع می نامیم.) پیمایش dfs را اجرا می‌کنیم و برای هر گره فاصله آن گره را از گره شروع(با فرض غیر جهت دار بودن همه‌ی یال‌ها) و نیز تعداد یال‌های در مسیر گره شروع تا گره کنونی که جهت آن‌ها باید تغییر کند را ذخیره می‌کنیم. این گونه یال ها را یال بازگشتی نام گذاری می‌کنیم. با استفاده از این پیمایش تعداد کل برعکس کردن جهت‌ها برای اینکه گره شروع به گره ریشه تبدیل شود(تعداد کل یال های بازگشتی برای گره شروع) را نیز پیدا می‌کنیم.

حال برای هر گره دیگر بدون انجام پیمایش dfs، می‌توانیم تعداد کل یال‌های بازگشتی را به صورت زیر محاسبه کنیم. فرض کنید می‌خواهیم تعداد کل یال‌های بازگشتی را برای گره شماره i بدست آوریم:

دو دسته یال بازگشتی خواهیم داشت. دسته‌ی اول یال‌های بازگشتی هستند که در مسیر یکتای بین گره شروع و گره شماره i وجود دارند که تعداد آن‌ها به وضوح برابر است با تعداد کل یال‌های بین گره شروع و گره i ، منهای تعداد یال‌های بازگشتی مسیر گره شروع به گره i که این مقدار را در بخش قبل محاسبه کردیم. در نتیجه:

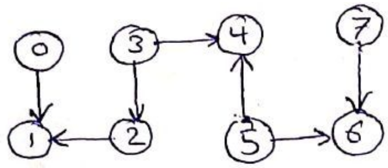
تعداد یال‌های بازگشتی دسته اول = فاصله‌ی گره i از گره شروع - تعداد یال‌های بازگشتی مسیر از گره شروع به گره i

دسته دوم یال‌های بازگشتی هستند که در مسیر یکتای بین گره شروع و گره شماره i قرار ندارند که به وضوح این یال‌های بازگشتی برای گره شروع و رأس i مشترک هستند پس تعداد آن‌ها به صورت زیر بدست می‌آید:

تعداد یال‌های بازگشتی دسته دوم = تعداد کل یال های بازگشتی برای گره شروع - تعداد یال‌های بازگشتی از گره شروع به گره i

تعداد کل یال‌های بازگشتی برای گره i برابر است با جمع یال های این دو دسته. کافیست این مقدار را برای هر گره دیگر بدست آوریم و در نهایت مقدار کمینه‌ی این اعداد جواب سوال خواهد بود.

مثال سوال۶:



گره دلخواه برای شروع را گره ۰ فرض می‌کنیم

پس از یکبار dfs با شروع از ۰ داریم

شماره گره	فاصله از رأس شروع	تعداد برعکس گردان یال‌های
0	0	0
1	1	0
2	2	1
3	3	2
4	4	2
5	5	3
6	6	3
7	7	4

مثلا در رأس شماره‌ی 3 فاصله از گره اولیه برابر 3 است و جهت دو یال ①←② و ②←③ باید برعکس شود تا بتوان از رأس ۰ به رأس 3 رسید.

با استفاده از این یکبار dfs که زده شده می‌دانیم جهت ۴ یال در کل باید برعکس شود

①←②
②←③
④←⑤
⑥←⑦

حال می‌خواهیم بدانیم برای رأس های دیگر چند یال باید برعکس شود تا بتوانند به رأس ریشه تبدیل شوند. برای مثال برای رأس شماره‌ی 5 داریم:

تعداد این گونه یال ها در رأس 5 تا 7 + تعداد این گونه یال ها در مسیر مسیر راس 5 تا 1 = تعداد یال هایی که جهت آن ها باید برعکس شود تا رأس 5 به رأس ریشه تبدیل شود

تعداد یال های بازگشتی از ۰ به 5 − تعداد کل برعکس گردن ها از ۰ به 7

فاصله‌ی رأس 5 تا رأس ۰ − تعداد یال های بازگشتی از ۰ به 5

5 − 3 = 2   4 − 3 = 1

پس در کل 3 + 1 یال باید تغییر جهت یابند تا رأس 5 به ریشه تبدیل شود.

به همین ترتیب برای همه‌ی رأس ها این مقدار را بدست می‌آوریم و کمینه را پیدا می‌کنیم.