# پاسخنامه تمرین شماره‌ی دو

دانشکده مهندسی برق و کامپیوتر | استاد: دکتر هشام فیلی | ساختمان های داده و الگوریتم

پاییز ۱۴۰۰

طراح: بهار افشار

---

۱. تبدیلات خواسته شده را انجام دهید و وضعیت پشته عملوند و عملگر را در نقاط مشخص شده با پرچم قرمز را نشان دهید.

(اعداد تک رقمی هستند.)

الف) تبدیل عبارت میانوندی زیر به عبارت پسوندی معادلش.

1+3/5 *(4-(6/(8+ 4)* 7))

پاسخ. (منبع)

از استک برای نگهداری وضعیت عملگرها استفاده می‌کنیم. (الگوریتم این تبدیل در لینک بالا آمده است)

1+3/5

| \ |
|---|
| + |

رشته خروجی تا این لحظه:

1+3/5 ⌉ *(4-(6/(8+ ⌉

| |
|---|
| + |
| ( |
| / |
| ( |
| - |
| ( |
| * |
| + |

رشته خروجی تا این لحظه:

1 3 5 / 4 6 8 4

حاصل پسوندی معادل:

1 3 5 / 4 6 8 4 + / 7 * - * +

ب) تبدیل عبارت پسوندی زیر به عبارت پیشوندی معادلش.

5 4 * 3 ⌉ 7 / + ⌉ 6 4 + 3 ⌉ * -

پاسخ.

5 4 * 3 ⌉

| |
|---|
| 3 |
| *54 |

5 4 * 3 ⌉ 7 / + ⌉

| |
|---|
| +*54/37 |

5 4 * 3 | 7 / + | 6 4 + 3

| |
|---|
| 3 |
| +64 |
| +*54/37 |

حاصل پیشوندی معادل:

-+*54/37*+643

۲. جایگشت پشته‌ای، به جایگشتی می‌گویند که با اعمال عملیات‌های پشته (push/pop) بر روی یک مجموعه بدست می‌آید. دو صف از اعداد صحیح به شما داده شده است. الگوریتمی ارائه کنید تا مشخص کند آیا صف دوم یک جایگشت پشته‌ای از صف اول می‌باشد یا خیر. توجه کنید که تنها اجازه dequeue() از صف اول و enqueue() به صف دوم را دارید.

مثال:

| | |
|---|---|
| آرایش ابتدایی | 1, 2, 3 |
| آرایش پایانی | 2, 1, 3 |
| خروجی | بله |
| پروسه انجام | Push 1 from input to stack.<br>Push 2 from input to stack.<br>Pop 2 from stack to output.<br>Pop 1 from stack to output.<br>Push 3 from input to stack.<br>Pop 3 from stack to output. |

پاسخ. (منبع)

باید سعی کنیم تا با استک صف ورودی را به صف خروجی تبدیل کنیم. الگوریتم پیشنهادی برای این مسئله به صورت زیر است.

دو قدم زیر را تا زمانی که صف ورودی خالی شود انجام می‌دهیم. در نهایت اگر هم صف ورودی هم استک خالی بودند امکان تبدیل صف ورودی به خروجی وجود دارد در غیر این صورت این صف جایگشت‌پذیر نیست.

۱. از صف ورودی به صورت متوالی عناصر را خارج می‌کنیم اگر این عنصر برابر با عنصر سر صف خروجی بود قدم دوم را اجرا می‌کنیم. در غیر اینصورت آن را به استک پوش می‌کنیم.

۲. در صورتی که به عنصری رسیدیم که برابر با عنصر سر صف خروجی است از هر دو صف ورودی و خروجی یک عنصر dequeue() می‌کنیم. حال عنصر سر استک را با عنصر سر صف خروجی مقایسه می‌کنیم. اگر برابر بود از هر دو این عنصر را خارج می‌کنیم در غیر این صورت قدم اول را تکرار می‌کنیم.

برای فهم بیشتر کد این الگوریتم در تصویر زیر قابل مشاهده است.

```
def checkStackPermutation(Input, Output, n):

    tempStack = Stack()
    while (not Input.isEmpty()):
        ele = Input.dequeue()
        if (ele == output.top()):
            output.dequeue()
            while (not tempStack.isEmpty()):
                if (tempStack.top() == output.top()):
                    tempStack.pop()
                    output.dequeue()
                else:
                    break
        else:
            tempStack.append(ele)

    return (Input.isEmpty() and tempStack.isEmpty())
```

۳. یک لیست دو طرفه پیوندی مرتب شده به شما داده شده است. الگوریتمی ارائه کنید که در کمترین زمان ممکن تمام سه تایی هایی که مجموع آنها برابر یک عدد مشخص است را بدست آورد.

مثال:

| | |
|---|---|
| لیست پیوندی | 1<->2<->3<->5<->6<->9 |
| مجموع خواسته شده | 13 |
| سه‌تایی های یافت‌شده | (2,5,6), (1,3,9) |

پاسخ. (منبع)

برای اینکه بهینه‌ترین الگوریتم ممکن را داشته باشیم به این شکل عمل می‌کنیم. مجموع خواسته شده را با نام givenSum نشان می‌دهیم. از ابتدای لیست شروع به پیمایش می‌کنیم و به ازای هر نود x تابعی را صدا می‌کنیم که تمام جفت‌هایی که مجموعی برابر با (givenSum – x) داشته باشند را پیدا کند. این تابع برای این کار به دو پوینتر ابتدا و انتهای لیست نیاز دارد. از آنجایی که نمی‌خواهیم سه‌تایی های تکراری پیدا کنیم پوینتر ابتدایی را برابر پوینتر عنصر بعدی x قرار می‌دهیم.

حال به توضیح الگوریتم یافتن دوتایی‌ها با مجموع مشخص می‌پردازیم.

همانطور که قبلا گفته شد این الگوریتم از دو پوینتر که به ابتدا و انتهای لیست اشاره می‌کنند استفاده می‌کند.

اگر مجموع مقدار نودهای پوینتر اول و دوم بیشتر از مجموع خواسته شده باشد، پوینتر دوم را یکی به عقب میاوریم. و اگر کمتر باشد پوینتر اول را یکی به جلو می‌بریم. هر زمان نیز که این مجموع برابر با مجموع خواسته شده شود دو نود را در جایی ذخیره می‌کنیم و پوینتر اول را یکی به جلو و دومی را یکی به عقب می‌بریم. این عملیات تا زمانی ادامه میابد که دو پوینتر برابر شوند یا اولی از دومی بگذرد. هزینه زمانی این الگوریتم O(n) می باشد.

پس هزینه زمانی کل عملیات یافتن سه تایی ها $o(n^2)$ و هزینه حافظه آن o(1) خواهد بود.

برای فهم بیشتر کد زیر ارائه شده است.

```
# Function to count pairs whose sum
# equal to given 'value'
def findPairs(first, second, value):
    pairs = []

    while (first != None and second != None and
        first != second and second.next != first):

        # Pair found
        if ((first.data + second.data) == value):
            pairs.append((first.data,second.data))

            # Move first in forward direction
            first = first.next
            # Move second in backward direction
            second = second.prev

        # If sum is greater than 'value' move second in backward direction
        elif ((first.data + second.data) > value):
            second = second.prev

        # Else move first in forward direction
        else:
            first = first.next

    return pairs
```

```
# Function to count triplets in a sorted
# doubly linked list whose sum is equal
# to a given value 'x'
def findTriplets(head, x):
    triplets = []
    if (head == None):
        return 0

    current, first, last = head, None, None
    count = 0

    # Get pointer to the last node of the doubly linked list
    last = head
    while (last.next != None):
        last = last.next

    while current != None:
        first = current.next
        # find pairs with sum(x - current.data) in
        # the range first to last and add it to the triplets
        pairs = countPairs(first, last, x - current.data)
        x_triplets = [(current.data,p[0],p[1]) for p in pairs]
        triplets.extend(x_triplets)
        current = current.next

    return triplets
```

۴. در مرحله آخر بازی مرکب، n بازیکن در یک دایره ایستاده‌اند. در ابتدای بازی عدد k از طرف بازیگردان اعلام می‌شود و سپس بازی از جایگاه نفر اول در این دایره آغاز می‌شود و در همان جهت ادامه پیدا می‌کند. نفر اول فردی را که در فاصلهk نفر مقابل او قرار دارد را از بازی حذف می‌کند سپس نفر بعدی فرد حذف شده بازی را ادامه می‌دهد. این عملیات آنقدر ادامه می‌یابد تا دایره کوچک و کوچکتر شود و فقط یک نفر باقی بماند که برنده بازی می‌شود. شما قصد دارید در این بازی شرکت کنید، اما ابتدا باید الگوریتمی ارائه دهید که با دانستن اینکه در هر مرحله فاصله بین نفر اول و نفری که قرار است توسط او حذف شود، جایی را در این صف دایره‌ای پیدا کنید تا در صورتی که در آنجا بایستید برنده بازی شوید. (راهنمایی: از یک لیست پیوندی دوری استفاده کنید.)

مثال:

در صورتی که ۵ نفر در صف حضور داشته باشند و عدد اعلام شده از سمت بازیگردان ۲ باشد، جایگاه نفر برنده، جایگاه چهارم است.

پاسخ. (منبع)

کافیست تا افراد موجود در این دایره را با نودهای یک لیست پیوندی دوری نمایش دهیم و تا زمانی که تعداد نودها برابر یک نشده، هربار به اندازه عدد اعلام شده یک شمارنده بذاریم تا به فرد بعدی برسیم که قرار است حذف شود. حال این نود را حذف می کنیم.

برای فهم بیشتر کد این سوال در تصویر زیر قابل مشاهده است.

ابتدا یک لیست پیوندی دوری از افراد  می سازیم.

```
def create_cyclic_ll(n):
    head = Node(1)
    prev = head
    for i in range(2, n + 1):
        prev.next = Node(i)
        prev = prev.next
    #connect last node to the head to make it a circle
    prev.next = head
```

حال الگوریتم را روی این لیست اجرا می کنیم.

Head نود ابتدایی لیست پیوندی دوری است.(فردی که مسابقه از آن شروع می‌شود.)

K عدد اعلام شده از طرف بازیگردان می‌باشد.

```
def getWinnerPosition(k, head):

    ptr1 = head
    ptr2 = head
    while (ptr1.next != ptr1):
        # Find k-th person
        count = 1
        while (count != m):
            ptr2 = ptr1
            ptr1 = ptr1.next
            count += 1

        #Remove the k-th person
        ptr2.next = ptr1.next
        # set ptr1 to next person
        ptr1 = ptr2.next

    print("Last person left standing is ", ptr1.data)
```

۵. می‌خواهیم تعدادی عملیات جدید برای پشته را تعریف کنیم.

الف) پشته‌ای را طراحی کنید تا عملیات getMin() در زمان O(1) و حافظه اضافی O(1) انجام دهد.

پاسخ. (منبع)

برای این منظور یک متغیر با نام min_element در نظر می‌گیریم تا عنصر کمینه را در آن ذخیره کنیم. در زمان صدا شدن getMin() این متغیر برگردانده می‌شود. حال باید عملیات های push و pop را مجددا تعریف کنیم.

عملیات push(x):

۱. اگر استک خالیست، x را به استک اضافه می‌کنیم و min_element را برابر با x قرار می‌دهیم.

۲. اگر استک خالی نیست باید x را با min_element مقایسه کنیم. دو حالت پیش می‌آید.

الف. اگر x از min_element بزرگتر باشد، x را به استک اضافه می‌کنیم.

ب. اگر x از min_element کوچکتر باشد، مقدار (2*x – min_element) را به استک اضافه کرده و مقدار min_element را برابر با x قرار می‌دهیم.

عملیات pop():

فرض می‌کنیم عنصری که قرار است pop شود y باشد. دو حالت پیش می‌آید.

۱. اگر y بزرگتر مساوی min_element باشد، عنصر کمینه در استک همچنان min_element خواهد بود.

۲. اگر y کوچکتر از min_element باشد، مقدار min_element را به (2*min_element – y) آپدیت می‌کنیم. بدین شکل مقدار قبلی کمینه قبل از افزوده شدن y را بدست می‌آوریم.

ب) پشته‌ای را تعریف کنید تا عملیات Reverse() (معکوس سازی عناصر پشته) را در زمان O(n) و حافظه اضافی O(1) انجام دهد.

پاسخ. (منبع)

برای این منظور استک خود را با استفاده از لیست پیوندی می‌سازیم. پس برای معکوس کردن آن کافیست تا یک بار کل استک را در زمان O(n) پیمایش کنیم و پوینتر next نودها را عوض کنیم طوری که به جای نود بعدی به نود قبلی اشاره کند.

ج) با استفاده از یک پشته موقت، عملیات Sort() صعودی را برای پشته تعریف کنید.

پاسخ. (منبع)

الگوریتم پیشنهادی به صورت زیر است.

۱. ابتدا یک استک موقت با نام tempStack می‌سازیم.

۲. تا زمانی که استک اولیه خالی نشده قدم‌های زیر را انجام می‌دهیم.

الف. یک عنصر را از استک اولیه پاپ می‌کنیم و در متغیر temp ذخیره می‌کنیم.

ب. تا زمانی که tempStack خالی نشده و سر استک tempStack از temp بزرگتر است، از استک اولیه پاپ می‌کنیم و به tempStack پوش می‌کنیم.

ج. temp را به tempStack پوش می‌کنیم.

در نهایت عناصر موجود در tempStack مرتب‌شده عناصر استک اولیه هستند.

۶. به شما دو عدد که هر رقم آن در یک نود از لیست پیوندی ذخیره شده است داده‌اند. با استفاده از یک پشته الگوریتمی ارائه دهید که اعمال جمع و تفریق را برای این دو عدد انجام دهد و نتیجه را در یک لیست پیوندی ذخیره کند(هر رقم در یک نود).

مثال:

| Number | Linked List Representation |
|---|---|
| 9857 | 9->8->5->7->null |
| 65 | 6->5->null |
| 9857+65 = 9922 | null<-0<-9<-9<-2<-2 |

پاسخ.

برای حل این سوال ابتدا لیست‌های پیوندی را پیمایش کرده و آن دو را در دو استک stackA و stackB ذخیره می‌کنیم.

```
def create_stack_from_ll(head):
    stack = Stack()
    while (head.next != None):
        stack.push(head.data)
        head = head.next
    return stack
```

عملیات پاپ را برای این استک‌ها به این صورت تغییر می‌دهیم که اگر استک خالی باشد عدد -۱ را برگرداند.

حال تا زمانی که هر دو استک خالی نشدند، از آنها عنصر پاپ می‌کنیم. توجه کنید که اگر مقدار پاپ شده -۱ باشد یعنی این استک خالیست و نباید در محاسبه جمع در نظر گرفته شود پس در این حالت مقدار آن را صفر می‌کنیم. در غیر اینصورت مجموع دو عنصر را حساب می‌کنیم. در صورتی که این مجموع بزرگتر از ۱۰ باشد یعنی عملیات جمع carry داشته. پس مقدار مجموع و carry باید آپدیت شوند. در نهایت به ازای هر مجموع نود لیست پیوندی مربوطه را با این مقدار می سازیم تا در پایان حلقه، لیست پیوندی حاصل جمع ساخته شده باشد.

برای فهم بیشتر کد این مسئله در تصویر زیر آورده شده.

```
def add_stacks(stackA, stackB):
    carry = 0
    # create dummy node as head
    head = Node(-1)
    prev = head.next
    while !(stackA.isEmpty() and stackB.isEmpty()):
        a_element = stackA.pop()
        b_element = stackB.pop()

        if a_element == -1:
            a_element = 0
        if b_element == -1:
            b_element = 0
        value = a_element + b_element + carry

        if value >= 10:
            value = value % 10
            carry = 1

        # create node to store result
        prev.next = Node(value)
        prev = prev.next

    prev.next = None
    return head
```

## نکات تکمیلی

- در صورت وجود ابهام در مورد سوالات می‌توانید از طریق ایمیل با من در ارتباط باشید.
- دقت فرمایید که پاسخ سوال‌ها یکتا نیست و به دیگر پاسخ‌های صحیح نیز نمره تعلق می‌گیرد.

موفق باشید.