پاسخ تمرین شماره ۲

دانشکده مهندسی برق و کامپیوتر

ساختمان داده - بهار ۱۳۹۹

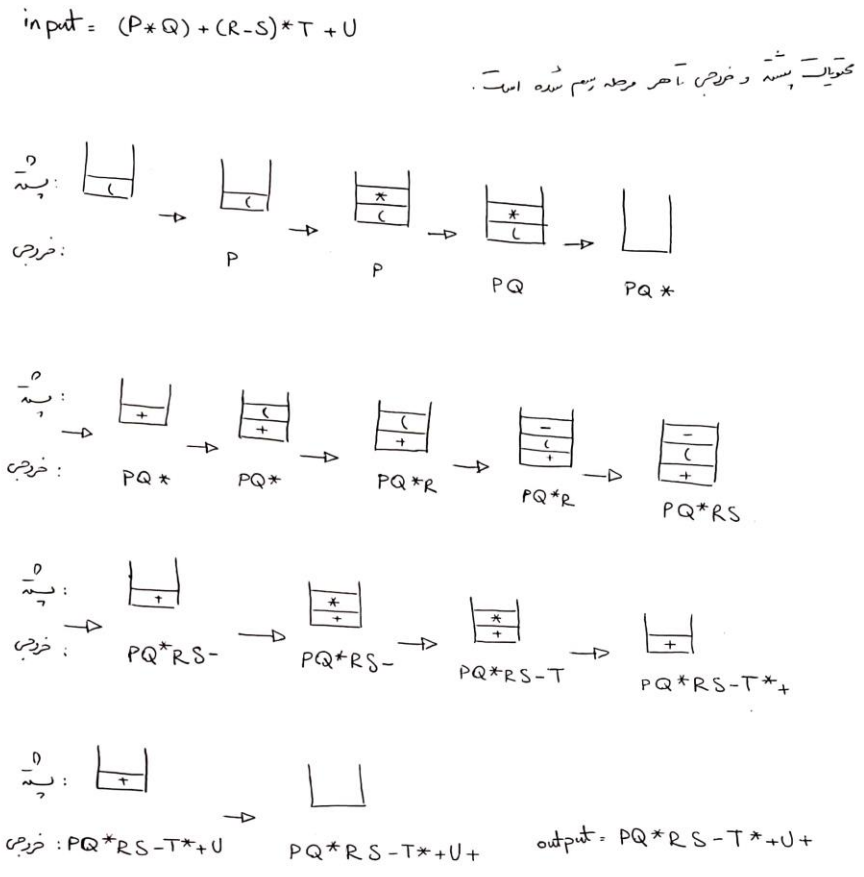
استاد : فتحیه فقیه

مسئول تمرین : رستا تدین
rasta.tadayon1378@gmail.com

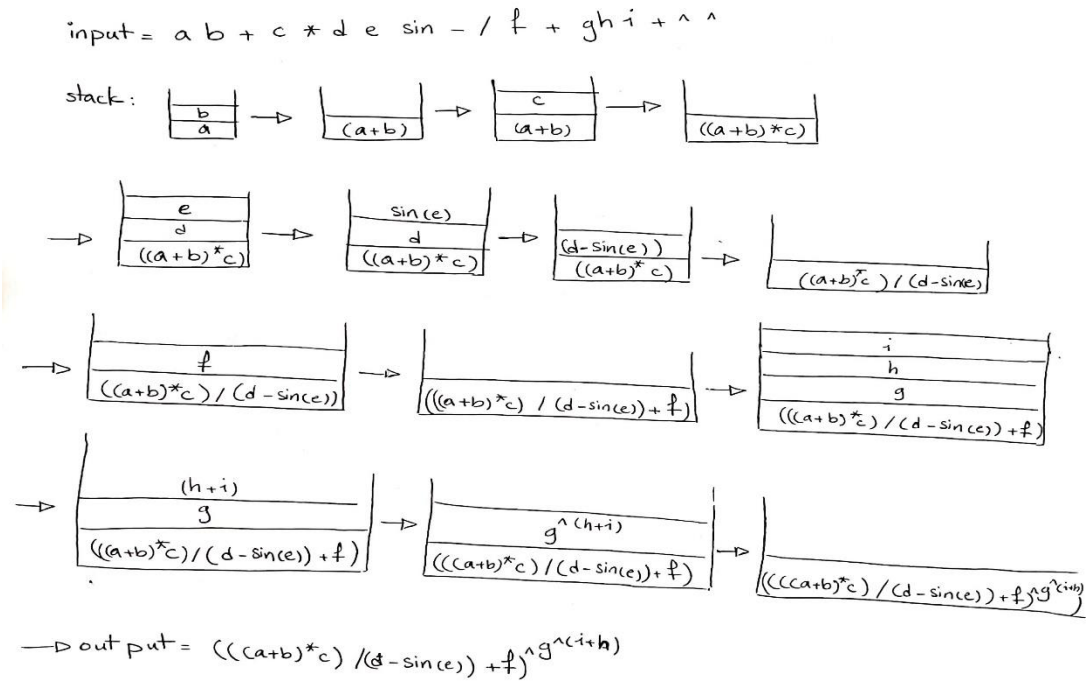
---

۱. مقادیر خواسته شده به این ترتیب خواهند بود:

Postfix = P Q * R S – T * + U +



Infix = ((a+b)*c/(d-sin(e)) + f)^(g^(i+h))



۲. برای حل این سوال از ۲ صف استفاده می‌کنیم.

در صف اول ایندکس‌های ۰ های آرایه را به ترتیب از راست به چپ ذخیره می‌کند. صف دوم ایندکس‌های ۱ های آرایه را به ترتیب از راست به چپ ذخیره می‌کند. (ایندکس راست ترین ۰ یا ۱ سر صف هستند) حال در صورت آمدن دستور اول از سر صف اول dequeue کرده، ایندکس مربوط از آرایه را حذف می‌کنیم و مقدار محتوی آن ایندکس آرایه را چاپ می‌کنیم. در صورت آمدن دستور دوم، از صف دوم dequeue کرده و مراحل قبل را انجام می‌دهیم. در صورت آمدن دستور سوم، سر هر دو صف را مقایسه کرده و ایندکس بیشتر را از صف مربوطه dequeue و از آرایه حذف و چاپ می‌کنیم.

می‌توانید از پشته هم برای حل سوال استفاده کنید، آرایه را پیمایش کرده و به ترتیب ایندکس‌های ۰ و ۱ را در پشته های متناظر push کنید. به این ترتیب ایندکس راست ترین ۰ و ۱ در بالای پشته‌ها قرار خواهند گرفت. سپس مراحل بالا را دوباره به جای صف برای پشته تکرار کنید.

۳ . برای حل این سوال در اوردر زمانی کمتر از $O(n^2)$از پشته کمک می‌گیریم.

ابتدا آرایه ورودی را بر اساس زمان شروع هر بازه به صورت صعودی مرتب می‌کنیم.

سپس الگوریتم زیر را انجام می‌دهیم:

۱) اولین بازه‌ی آرایه را در پشته push می‌کنیم. سپس روی بقیه بازه‌ها حلقه زیر را اجرا می‌کنیم.

۲,۱) برای بازه‌های بعدی در صورتی که با بازه top پشته تداخل نداشته باشند آنها را در پشته push می‌کنیم.

۲,۲) اگر بازه بعدی با بازه سر پشته تداخل داشت و زمان خاتمه آن بازه بیشتر از زمان خاتمه بازه top پشته بود زمان خاتمه سر پشته را برابر با زمان خاتمه آن بازه قرار می‌دهیم.

۳) زمانی که این حلقه تمام شود، محتویات داخل پشته شامل بازه‌هایی هستند که با یکدیگر تداخل ندارند و خروجی برنامه ما هستند.

این الگوریتم به دلیل مرتب سازی اولیه پیچیدگی زمانی nlogn دارد.

۴. الف) مشخص است عددی که بر ۲ و ۵ بخش پذیر باشد به طور حتم یکانی برابر ۰ خواهد داشت بنابراین بزرگترین عدد بخش پذیر بر ۳ را بدست می‌آوریم و اگر آرایه شامل ۰ نبود در خروجی اعلام می‌کنیم که نمی‌توان با آرایه ورودی داده شده، شروط مسئله را برآورده کرد. اعدادی بر ۳ بخش پذیر هستند که مجموع ارقام آنها بر ۳ بخش پذیر باشد. حال برای بدست آوردن بزرگترین عدد بخش پذیر بر ۳ عناصر آرایه را به صورت صعودی مرتب می‌کنیم. می‌دانیم اعداد می‌توانند بر ۳ باقیمانده هایی برابر ۰،۱ و ۲ داشته باشند. برای هر یک از این مقادیر یک صف به نام‌های queue0, queue1,queue2 در نظر می‌گیریم و روی آرایه مرتب شده پیمایش میکنیم و اعداد را در صف مناسبشان enqueue می‌کنیم.

سپس مجموع تمام ارقام آرایه را بدست می‌آوریم. ۳ حالت برای این مجموع پیش می‌آید:

۱) بر ۳ بخش پذیر است. تمام اعضای ۳ صف را از آنها خارج می‌کنیم، به صورت نزولی مرتب می‌کنیم و به عنوان خروجی می‌دهیم.

۲) بر ۳ باقیمانده ای برابر ۱ دارد. در این حالت اگر queue1 حداقل یک عضو داشت، عضو اول آن را dequeue می‌کنیم در غیر این صورت اگر queue2 حداقل ۲ عضو داشت، دو عضو از آن dequeue می‌کنیم. اگر هیچ یک از شرایط گفته شده برقرار نبود، در خروج اعلام می‌کنیم که نمی‌توان شرایط مسئله را برقرار کرد.

۳) بر ۳ باقیمانده برابر ۲ دارد. در این حالت اگر queue2 حداقل یک عضو داشت، عضو اول آن را dequeue می‌کنیم در غیر این صورت اگر queue1 حداقل ۲ عضو داشت، دو عضو از آن dequeue می‌کنیم. اگر هیچ یک از شرایط گفته شده برقرار نبود، در خروجی اعلام می‌کنیم که نمی‌توان شرایط مسئله را برقرار کرد.

۴) نهایتا تمام اعضای صف‌ها را در یک آرایه موقتی dequeue می‌کنیم و به صورت نزولی مرتب می‌کنیم و به عنوان خروجی می‌دهیم.

ب) برای حل مسئله با پیچیدگی زمانی O(n) رویکردی تقریبا مانند قسمت قبل داریم با این تفاوت که با توجه به این که حدود اعداد را داریم و می‌دانیم اعداد آرایه شامل اعداد صحیح بین ۰ تا ۹ هستند می‌توانیم برای مرتب کردن آنها ۱۰ صف در نظر بگیریم. سپس روی اعداد آرایه پیمایش کنیم و هر عدد را در صف مخصوص به خود قرار دهیم یعنی در صورت مشاهده عدد ۱ آن را در صف ۱ و عدد ۲ آن را در صف ۲ و به همین ترتیب قرار دهیم. (اگر صف ۰ خالی بود عدم امکان برآورده کردن شرایط مسئله را در خروجی اعلام می‌کنیم.) سپس از بزرگترین صف یعنی صف ۹ شروع کرده و اعداد داخل آن ها را در ۳ صف مربوط به باقی مانده ها بر ۳ تعریف شده در قسمت قبل enqueue می‌کنیم به این ترتیب هر یک از ۳ صف مرتب شده خواهند بود. سپس الگوریتم قسمت قبل را با داشتن ۳ صف مربوط به مقادیر باقیمانده دوباره اجرا می‌کنیم. در نهایت پس از اجرای الگوریتم محتوی صف‌ها را در یک آرایه موقتی dequeue می کنیم و برای مرتب کردن به صورت نزولی دوباره از روش گفته شده در همین قسمت استفاده می‌کنیم. آرایه مرتب شده را به عنوان خروجی می‌دهیم.

۵. الف)

دو اشاره‌گر در نظر می‌گیریم که اولی به ابتدای لیست پیوندی دو طرفه و کوچکترین عضو آن اشاره می‌کند و دومی به انتهای آن و بزرگترین عضو آن اشاره می‌کند.

اگر مجموع مقادیری که اشاره‌گرها به آن‌ها اشاره می‌کنند کمتر از x بود اشاره‌گر اول را جلو می‌بریم و اگر بیشتر از x بود، اشاره‌گر دوم را به عقب برمی‌گردانیم. اگر مجموع برابر x شد به خروجی اضافه می‌کنیم و اشاره‌گر اول را یکی جلو برده و اشاره‌گر دوم را یکی به عقب برمی‌گردانیم.

هنگامی که هر کدام از اشاره‌گر ها NULL شدند یا از یکدیگر عبور کردند (مقدار اشاره‌گر اول از دوم بیشتر شود) یا با یکدیگر برابر شدند الگوریتم پایان می‌یابد.

ب) بله. برای حل کردن این مسئله با n اشاره‌گر اضافه می‌توان لیست پیوندی یک طرفه را با یک بار پیمایش به لیست پیوندی دو طرفه تبدیل کرد و مانند قسمت قبل مسئله را حل کرد. برای حل مسئله بدون استفاده از فضای اضافه کافی است این لیست پیوندی را به لیست پیوندی xor تبدیل کرد که با یک بار پیمایش لیست قابل انجام است.

می‌توانید در رابطه با این لیست پیوندی بیشتر در این لینک مطالعه کنید: XOR Linked List

ج) برای پیاده سازی صف کافی است ۲ اشاره‌گر به لیست پیوندی اضافه کنیم. اشاره‌گر اول به ابتدای صف و دیگری به انتهای آن اشاره می‌کند.

برای پیاده سازی enqueue: یک node به انتهای لیست پیوندی (بعد از جایی که اشاره‌گر دوم به آن اشاره می‌کند) اضافه می‌کنیم و اشاره‌گر دوم به نود جدید اضافه شده اشاره خواهد کرد.

برای پیاده سازی dequeue: node ای که اشاره‌گر اول به آن اشاره می‌کند را حذف می‌کنیم و اشاره‌گر اول را به عنصر بعد منتقل می‌کنیم.

۶. برای حل این سوال از ۲ آرایه و یک پشته استفاده می‌کنیم. آرایه اول برای نگهداری تعداد تکرار مقادیر تعداد مبتلایان استفاده می‌شود. در واقع از مقادیر آرایه ورودی داده شده به عنوان index این آرایه و تعداد تکرار همین مقدار را، داده موجود در خانه مربوط به آن در نظر می‌گیریم. این آرایه برای مثالی که در صورت سوال آورده شده به صورت زیر خواهد بود.

A = [2, 2, 3, 4, 5, 3, 2]

تعداد تکرارهای هر عضو از آرایه = 3,3,2,1,1,2,3

Freq = [0, 0, 3, 2, 1, 1]

حال الگوریتم زیر را اجرا می‌کنیم:

۱- آرایه A را از راست به چپ پیمایش می‌کنیم.

۲- در هر مرحله به سر پشته توجه می‌کنیم. اگر پشته خالی بود خانه متناظر را در خروجی ۱- قرار می‌دهیم.

۳- در صورتی که پشته خالی نبود:

۳,۱- تعداد تکرار (مقدار موجود در آرایه freq) که سر پشته به آن اشاره می‌کند را با تعداد تکرار فعلی مقایسه می‌کنیم. در صورتی که تکرار عنصر سر پشته بیشتر از تکرار عنصر فعلی باشد، عنصری که سر پشته به آن اشاره می‌کند را در خانه متناظر خروجی قرار می‌دهیم.

۳,۲- اگر مقدار تکرار عنصری که سر پشته به آن اشاره می‌کند کمتر یا مساوی تعداد تکرار عنصر فعلی بود عناصر پشته را تا زمانی که تعداد تکرار عنصر سر پشته بیشتر از تعداد تکرار عنصر فعلی شود pop می‌کنیم. اگر پشته خالی نشد، عنصری که سر پشته به آن اشاره می‌کند را در خانه متناظر خروجی قرار می‌دهیم و در غیر این صورت مقدار -۱ را در آن خانه قرار می‌دهیم.

۴- در پایان هر تکرار حلقه ایندکس عنصر کنونی را پشته push می‌کنیم.

شبه کد این سوال به این صورت است:

```
def question6(a, n):
    if (n <= 0):
        return []
    stack = []
    freq = [0]*(max(a)+1)
    for i in a:
        freq[i] += 1 #making the frequency array
    res = [0]*n

    for i in range(n-1,-1,-1):    #reading the input array from right to left
        if (len(stack) == 0):
            res[i] = -1
        else:
            if (freq[a[stack[-1]]] > freq[a[i]]):
                res[i] = a[stack[-1]]
            else:
                while (len(stack) > 0 and freq[a[stack[-1]]] <= freq[a[i]]):
                    stack.pop()
            if (len(stack) == 0):
                res[i] = -1
            else:
                res[i] = a[stack[-1]]
        stack.append(i)
    return res
```

اوردر زمانی این الگوریتم خطی است زیرا هر ایندکس حداکثر یک بار در پشته push و یک بار از آن pop می‌شود.