حل تمرین شماره ۲

Arrays – Stack – Queue – Expressions – Linked Lists

رستا تدین طهماسبی
rasta.tadayon1378@gmail.com

STACK

Stack یک ساختمان داده LIFO(Last In First Out) است. توابع اصلی پیاده سازی شده برای این ساختمان داده شامل push, pop, top می‌شوند که فرض می‌شود که همگی آنها با پیاده سازی درست در O(1) انجام می‌پذیرند.

- **push**: عنصر جدیدی را در ابتدای stack اضافه می‌کند.
- **pop**: آخرین عنصر اضافه شده به stack را حذف کرده و به عنوان خروجی می‌دهد.
- **top**: آخرین عنصر اضافه شده را به عنوان خروجی می‌دهد اما آن را از stack حذف نمی‌کند.

✓از ساختمان داده stack هنگام فراخوانی توابع به صورت بازگشتی استفاده می‌شود. هنگام فراخوانی بازگشتی تابع، متغیرهای محلی و آدرس بازگشت در stack، push شده و هنگام بازگشت این متغیرها pop می‌شوند.

✓از stack برای انجام عملیات‌های محاسباتی نیز استفاده می‌شود که در اسلایدهای بعد به طور کامل توضیح داده خواهد شد.

QUEUE

Queue یک ساختمان داده FIFO(First In First Out) است. توابع پیاده سازی شده برای این ساختمان داده شامل enqueue و dequeue می‌شوند که با توجه به نوع پیاده سازی ساختمان داده صف و در صورت بهینه بودن آن در O(1) انجام پذیر هستند.

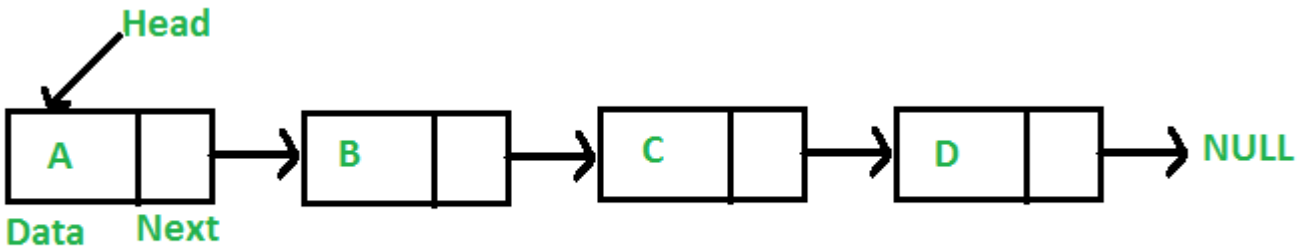
- **enqueue**: عنصر جدید را به انتهای صف اضافه می‌کند.
- **dequeue**: عنصر ابتدایی صف را از صف حذف کرده و به عنوان خروجی می‌دهد.

✓ از این ساختمان داده برای اولویت دادن به درخواست‌ها در سیستم‌های عامل استفاده می‌شود. درخواست‌ها در یک صف قرار گرفته و به ترتیب رسیدگی می‌شوند. (البته الگوریتم‌های بسیار پیچیده‌تری برای انتخاب درخواستی که در مرحله بعد اجرا می‌شود وجود دارد و FIFO تنها یکی از آنهاست.)

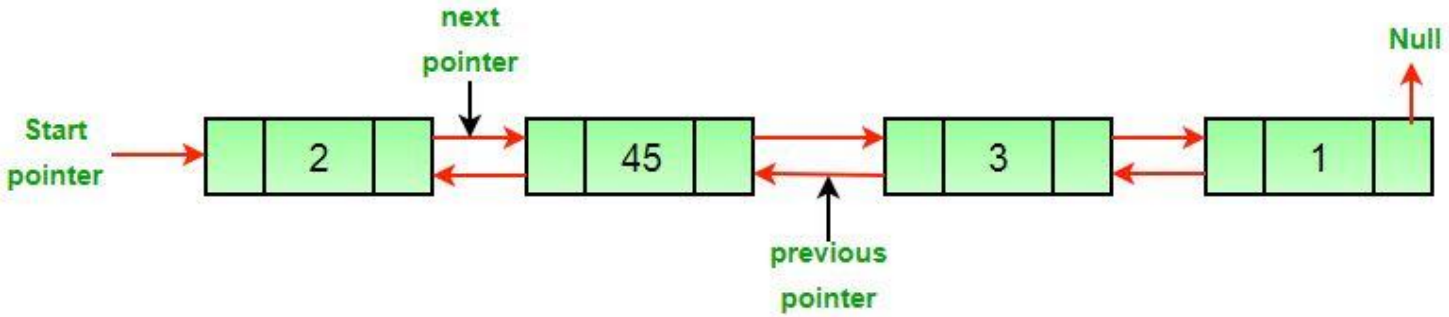
Linked Lists

یکی از روش‌های پیاده سازی ساختمان داده list استفاده از اشاره‌گر است که به آن linked list گفته می‌شود. Linked list ها انواع متفاوتی دارند که از آن‌ها می‌توان به linked list یک طرفه، linked list دو طرفه و linked list حلقوی اشاره کرد. ترتیب یک linked list وابسته به اشاره‌گرهای هر عنصر است و همانند دیگر لیست‌ها هر عنصر آن شامل key (یا data) است و علاوه بر آن شامل یک اشاره‌گر به عنصر بعد می‌شود.

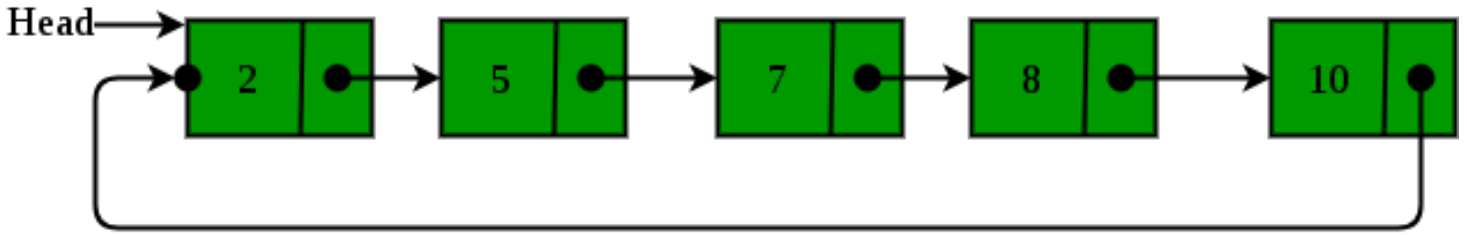
- Linked list دو طرفه: علاوه بر اشاره‌گر به عنصر بعد شامل یک اشاره‌گر به عنصر قبل است. اگر یک لینک لیست یک طرفه باشد، هزینه یافتن عنصر قبلی برابر O(n) است، در حالی که اگر ۲ طرفه باشد برابر O(1) است، این در حالی است که به ازای هر عنصر هزینه حافظه برای اشاره‌گر به عنصر قبل را متحمل می‌شویم بنابراین با تحمل هزینه حافظه بیشتر هزینه زمانی را کاهش می‌دهیم.
- Linked list حلقوی: در این لینک لیست آخرین عنصر یک اشاره‌گر به عنصر اول (که اشاره‌گر head به آن اشاره می‌کند) دارد.



لینک لیست یک طرفه



لینک لیست دو طرفه



لینک لیست حلقوی

عبارت‌های محاسباتی

عبارات محاسباتی را به ۳ طریق می‌توان نشان داد:

- Prefix (پیشوندی)
- Infix (میان‌وندی)
- Postfix (پسوندی)

روش میان‌وند روش متداول و راحت‌تری است اما به دلیل ابهاماتی که در اولویت انجام عملیات وجود دارد نیاز به پرانتز گذاری دارد، به همین دلیل عباراتی که به صورت پیشوندی و پسوندی نوشته می‌شوند، به دلیل عدم نیاز به پرانتز گذاری سریع‌تر ارزیابی می‌شوند.

همان طور که گفته شد برای تبدیل این عبارات به یکدیگر از ساختمان داده stack استفاده می‌شود.

تبدیل مستقیم پیشوندی به پسوندی

(می‌توانید یک عبارت پیشوندی را ابتدا به یک عبارت میان‌وندی تبدیل کنید و سپس عبارت میان‌وندی حاصل را به عبارت پسوندی تبدیل کنید اما این تبدیل‌ها پیچیده‌تر خواهند بود بنابراین الگوریتم تبدیل مستقیم اینجا آورده شده است.)

1. رشته را از راست به چپ (در جهت معکوس) بخوانید.
2. اگر المان خوانده شده عملوند است، آن را در پشته push کنید.
3. اگر المان خوانده شده عملگر است، ۲ مقدار را از پشته pop کنید و یک رشته به صورت زیر تولید کنید. سپس رشته تولید شده را در پشته push کنید.

   string = operand1 + operand2 + operator

4. تا تمام شدن رشته ادامه دهید. عنصر باقیمانده در پشته خروجی خواهد بود.

در اسلاید بعد با مشاهده یک مثال این تبدیل قابل فهم تر خواهد شد.

خانه‌های پشته با '|' از یکدیگر جدا شده‌اند.
دقت کنید رشته‌های ساخته شده در یک خانه پشته قرار می‌گیرند.

ورودی (پیشوندی)	پشته
* - A / - * B ^ C D E * F G **H**	H
* - A / - * B ^ C D E * F **G** H	H | G
* - A / - * B ^ C D E * **F** G H	H | G | F
* - A / - * B ^ C D E ***** F G H	H | FG*
* - A / - * B ^ C D **E** * F G H	H | FG* | E
* - A / - * B ^ C **D** E * F G H	H | FG* | E | D
* - A / - * B ^ **C** D E * F G H	H | FG* | E | D | C
* - A / - * B **^** C D E * F G H	H | FG* | E | CD^
* - A / - * **B** ^ C D E * F G H	H | FG* | E | CD^ | B
* - A / - ***** B ^ C D E * F G H	H | FG* | E | BCD^*
* - A / **-** * B ^ C D E * F G H	H | FG* | BCD^*E-
* - A **/** - * B ^ C D E * F G H	H | BCD^*E-FG*/
* - **A** / - * B ^ C D E * F G H	H | BCD^*E-FG*/ | A
* **-** A / - * B ^ C D E * F G H	H | ABCD^*E-FG*/-
***** - A / - * B ^ C D E * F G H	ABCD^*E-FG*/-H*

خروجی

مثال ۱
تابعی بازگشتی بنویسید که عناصر یک پشته از اعداد صحیح را به صورت صعودی مرتب کند ،یعنی سر پشته بزرگترین عنصر قرار گیرد.

ایده اصلی حل این مسئله نگه داشتن تمام مقادیر پشته در پشته مربوط به فراخوانی‌های بازگشتی است تا زمانی که خالی شود و سپس قرار دادن تمام مقادیر پشته به ترتیب صعودی در آن است.
حال برای برقرار کردن ترتیب صعودی اگر پشته خالی باشد صرفا المان داده شده در پشته push می‌شود. اگر پشته خالی نبود و top آن کمتر از المان داده شده بود باز هم در پشته push می‌شود.
حال اگر شرایط گفته شده برقرار نبود به این معناست که المان داده شده باید جایی در وسط پشته قرار بگیرد (با این فرض که پشته تا لحظه کنونی مرتب شده است و بزرگترین عنصر آن در top پشته قرار دارد.) حال برای قرار دادن المان در جای مناسب خود، یک عنصر از پشته pop می‌شود و دوباره تابع sortedInsert به صورت بازگشتی فراخوانی می‌شود و اگر شرایط گفته شده برای پشته باقیمانده برقرار باشد المان در پشته push می‌شود و با توجه به اینکه عنصر(یا عناصر) پاپ شده از یک پشته مرتب pop شده است پس از اتمام فراخوانی بازگشتی با ترتیب صحیح دوباره در پشته push می‌شوند و پس از اتمام فراخوانی‌های بازگشتی پشته داده شده مرتب است.

```
def sortedInsert(s , element):
    # Base case: Either stack is empty or newly inserted
    # item is greater than top (more than all existing)
    if len(s) == 0 or element > s[-1]:
        s.append(element)
        return
    else:

        # Remove the top item and recur
        temp = s.pop()
        sortedInsert(s, element)

        # Put back the top item removed earlier
        s.append(temp)

# Method to sort stack
def sortStack(s):

    # If stack is not empty
    if len(s) != 0:

        # Remove the top item
        temp = s.pop()

        # Sort remaining stack
        sortStack(s)
```

مثال ۲
با استفاده از یک پشته، عناصر لیست پیوندی را معکوس کنید.

کافی است به سادگی روی لیست پیوندی پیمایش کنید و عناصر آن را در پشته push کنید. سپس عناصر داخل پشته را pop کنید و لیست پیوندی را تشکیل دهید.

مثال ۳
با در دست داشتن یک صف از اعداد طبیعی ۱ تا n الگوریتمی طراحی کنید که چک کند که آیا امکان دارد عناصر این صف را با استفاده از یک پشته به صورت صعودی مرتب کرد و در صف دومی قرار داد یا خیر.
عملیات‌های مجاز شامل : push و pop از پشته – enqueue در صف دوم – dequeue از صف اول

صف دوم می‌تواند المان‌های ورودی اش را از پشته یا صف اول دریافت کند بنابراین عنصر مورد انتظار بعدی (که در ابتدا ۱ خواهد بود) باید به عنوان عنصر جلویی از صف ۱ یا عنصر بالایی پشته موجود باشد. بنابراین فرایند ساختن صف دوم را با قرار دادن عنصر مورد نظر بعدی برابر ۱ شروع می‌کنیم و چک می‌کنیم که آیا می‌توانیم عنصر مورد نظر را از بالای پشته یا جلوی صف دریافت کنیم یا خیر. اگر نتوانیم این المان را از هیچ یک از این ۲ طرق به دست آوریم، عنصر جلوی صف را dequeue کرده و در پشته push می‌کنیم.

دقت داشته باشید که محتویات پشته باید در هر لحظه مرتب شده باشند یعنی کوچکترین المان در سر پشته قرار داشته باشد چرا که اگر مقدار کوچک تر در لایه‌های پایینی‌تر پشته وجود داشته باشد باید زودتر از مقدار بزرگتر در صف دوم قرار بگیرد اما خارج کردن این مقدار کوچکتر از پشته بدون خارج کردن مقدار بزرگتر بالای پشته امکان پذیر نیست. بنابراین نمی‌توانیم در پشته مقداری بیشتر از سر پشته push کنیم.

حال الگوریتم اجرایی به این صورت خواهد بود:

1. مقدار مورد انتظار را برابر ۱ قرار می‌دهیم.
2. چک می‌کنیم که بالای پشته یا جلوی صف هیچ یک دارای این مقدار هستند یا خیر.
    1. اگر بله آن مقدار را از ساختمان داده مورد نظر خارج کرده و در صف ۲ قرار می‌دهیم سپس مقدار مورد انتظار را یک واحد افزایش می‌دهیم.
    2. در غیر این صورت مقدار جلوی صف را dequeue کرده و در پشتهpush می‌کنیم. اگر مقداری که از صف ۱ خارج شده بیشتر از مقدار سر پشته باشد در خروجی اعلام می‌کنیم که این کار قابل انجام نیست.
3. این مراحل را تا تمام شدن مقادیر موجود در صف۱ و پشته تکرار می‌کنیم و نهایتا در خروجی اعلام می‌کنیم که می‌توان این کار را انجام داد.

مثال ۴
در یک مهمانی با حضور n نفر، تنها یک فرد می‌تواند وجود داشته باشد که همه او را بشناسند. این شخص یک فرد مشهور است. اگر فرد مشهور در مهمانی حضور پیدا کند هیچ فرد دیگری را نمی‌شناسد. فرض کنید تابعی داریم که جواب سوال « آیا فرد A فرد B را می‌شناسد؟» را به ما می‌دهد، در این صورت با استفاده از پشته راه حلی پیدا کنید که با کمترین تعداد دفعات مطرح کردن پرسش در صورت حضور فرد مشهور او را شناسایی کند.

می دانیم اگر سوال « آیا فرد A فرد B را می‌شناسد؟» را مطرح کنیم ۲ جواب ممکن وجود دارد. اگر جواب بله باشد پس شخص A نمی تواند فرد مشهور باشد و حذف می‌شود، B می‌تواند فرد مشهور باشد که باید چک شود. اگر جواب خیر باشد، شخص A ممکن است مشهور باشد که باید چک شود و شخص B نمی‌تواند مشهور باشد و حذف می‌شود.

1. یک پشته در نظر می‌گیریم تمام افراد را در آن push می‌کنیم.
2. ۲ نفر را از بالای پشته pop کرده و سوال را میان آنها مطرح می‌کنیم و بر اساس جواب یکی از افراد را حذف می‌کنیم.
3. شخص باقیمانده را در پشته push می‌کنیم.
4. گام‌های ۲ و ۳ را تکرار می‌کنیم تا تنها یک شخص در پشته باقی مانده باشد.
5. چک میکنیم شخص باقیمانده بقیه را نشناسد و همه او را بشناسند

بنابراین $3*(n-1)$ بار پرسش مطرح می‌شود. $n-1$ بار اول برای مطرح کردن سوال در پشته است. $2*(n-1)$ بار بعدی برای گام ۵ ام است زیرا برای فرد باقی مانده در پشته به ازای هر شخص دیگر حاضر در مهمانی یک بار سوال برای اینکه مطمئن شویم همه او را می‌شناسند و یک بار برای اینکه مطئن شویم او کسی را نمی‌شناسد، مطرح می‌شود.

مثال ۵
فرض کنید به عنوان ورودی یک رشته از پرانتزهای باز و بسته به طول n به شما داده شده است. شبه کدی با پیچیدگی زمانی O(n) بنویسید که طول طولانی ترین زیر رشته متوازن را خروجی بدهد.

الگوریتم استفاده شده به این صورت است:

1. یک پشته خالی ایجاد کنید و -۱ را در آن push کنید. عنصر اول از پشته به عنوان پایه برای رشته معتبر بعدی استفاده می شود.
2. متغیر پاسخ را برابر ۰ قرار دهید.
3. ورودی را بخوانید و اگر برابر کاراکتر ‘)’ بود، ایندکس i را در پشته push کنید.
4. در غیر این صورت اگر برابر کاراکتر ‘(‘ بود:
   4.1. یک عنصر را از پشته pop کنید.
   4.2. اگر پشته خالی نبود طول رشته معتبر تا کنون را به این صورت محاسبه کنید:

   current length = current index – index of the top element of the stack

   اگر این مقدار بیشتر از مقدار متغیر پاسخ بود، آن را آپدیت می‌کنیم.
   4.3. اگر پشته خالی بود ایندکس فعلی را به عنوان پایه برای طول‌های بعدی در پشته push کنید.
5. متغیر پاسخ را به عنوان خروجی بیرون دهید.

شبه کد سوال ۵:

```
def findMaxLen(string):
    n = len(string)

    stk = []
    stk.append(-1)

    result = 0
      for i in range(n):

        if string[i] == '(':
            stk.append(i)

        else:

            stk.pop()

            if len(stk) != 0:
                result = max(result, i - stk[len(stk)-1])


            else:
                stk.append(i)

    return result
```