به نام خدا

درس ساختمان داده ها و الگوریتم ها

پاسخنامه ی تمرین سری دوم

آذر ۹۹

سوال۱) یک صف (Queue) را با استفاده از دقیقا دو پشته (Stack) شبیه سازی کنید.

استک ها را S1,S2 مینامیم. فرض کنید از استک S1 برای ذخیره ی داده ها و از S2 به عنوان فضای کمکی استفاده میکنیم.

راه اول:

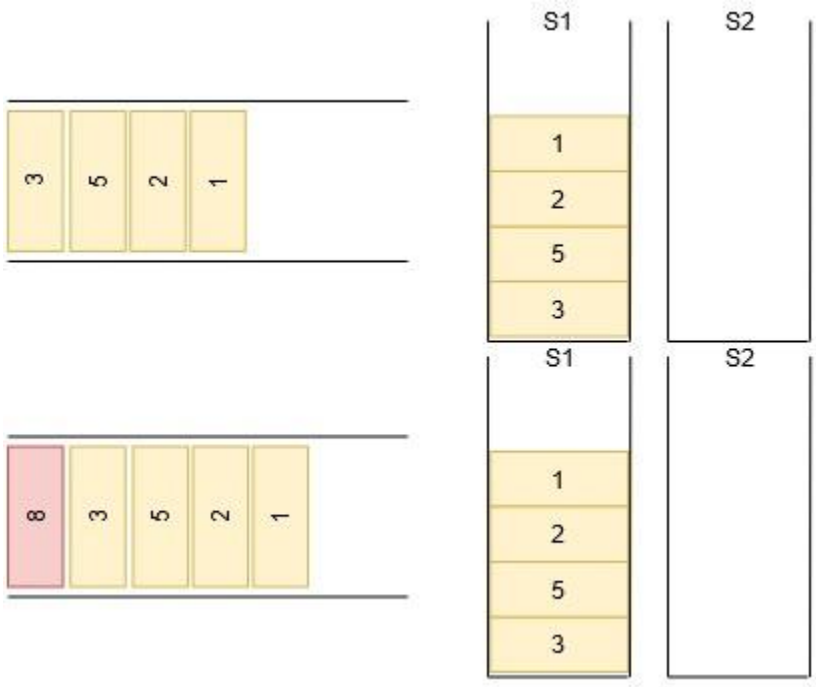
Enqueue -> O(n)

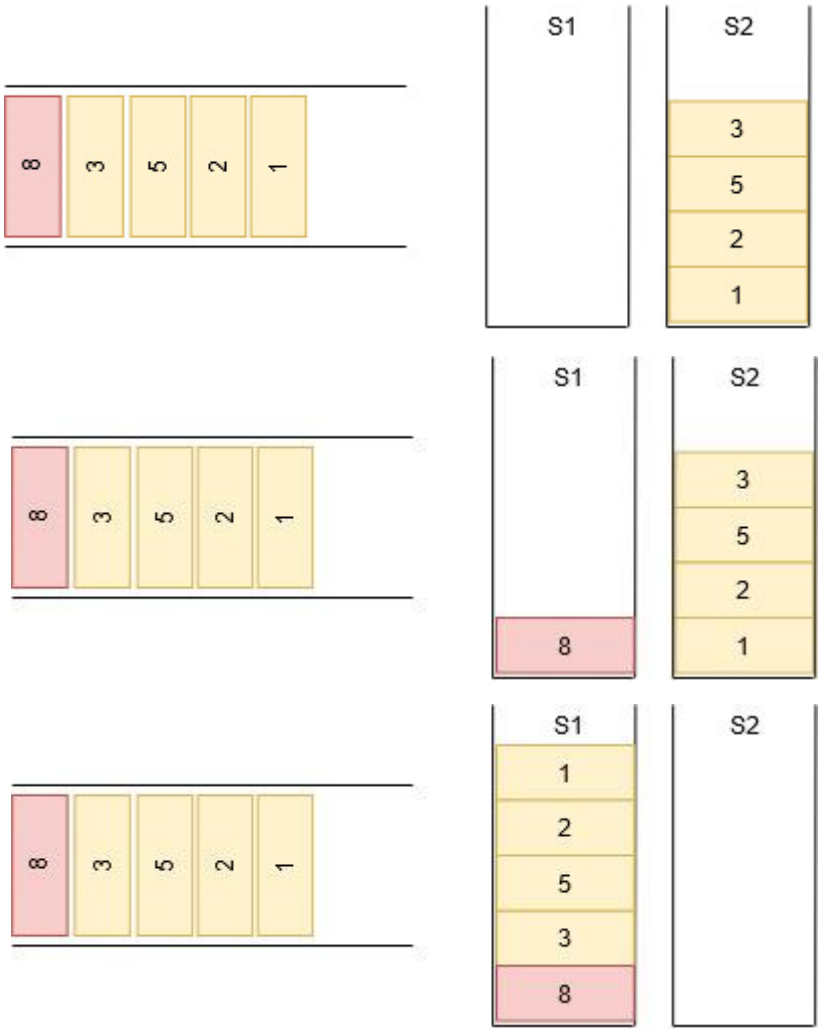
Dequeue -> O(1)

برای Enqueue کردن عنصر x، ابتدا تمام اعضای S1 را pop می کنیم و در S2 ، push می کنیم. حال x را در S1، push میکنیم و سپس تمامی اعضایی که در S2 ریخته شده بودند را pop کرده و دوباره در S1 ، push میکنیم.

برای Dequeue کردن صرفا از S1، pop می کنیم.

مثال تصویری برای Enqueue کردن عدد ۸ در صف:





و برای Dequeue کردن نیز به سادگی عنصر 1 را از S1، pop میکنیم.

راه دوم:

Enqueue -> O(1)

Dequeue -> O(n)

برای Enqueue کردن عنصر x آن را مستقیم در S1، push می کنیم.

برای Dequeue کردن، تمامی عناصر موجود در S1 را pop کرده و در S2، push می کنیم. آخرین عنصری که pop کرده ایم همان عنصری است که باید Dequeue شود. آن را می کنیم. حال دوباره تمامی عناصر را به S1 برمیگردانیم.

راه سوم:

Enqueue -> O(1)

Dequeue -> O(n)

عملیات Enqueue همانند راه دوم می باشد و عنصر x آن را مستقیم در S1، push می کنیم.

عملیات Dequeue:

ابتدا چک میکنیم که S2 خالی است یا خیر. اگر خالی نبود مستقیم از آن pop می کنیم. اگر خالی بود تمامی عناصر S1 را همانند راه دوم به S2 منتقل می کنیم و آخرین عضو را pop می کنیم.

در این راه حل نیازی نیست دوباره عناصر را از S2 به S1 برگردانیم. علت آن نیز این است که هر سری که میخواهیم Dequeue کنیم، S2 را چک می کنیم. در واقع این راه سوم بهینه شده ی راه دوم است.

سوال۲) الگوریتم Inorder_Tree_walk را به صورت یک الگوریتم غیر بازگشتی بیان کنید.

برای این کار نیاز داریم از یک ساختمان داده استفاده کنیم که اعضای پیمایش شده را در آن ذخیره کنیم تا عضوی را چندبار پیمایش نکنیم. (اگر از صف استفاده کنیم ترتیب خروج اعضا از آن معادل ترتیب پیمایش آنها میشود)

```
Cur = head
Explored = []
While cur != null:
    If cur.left != null and cur.left.value not exist in explored:
        Cur = cur.left
    Else if cur.value not exist in explored:
        Explored.add(cur.value)
        Print(cur.value)
    Else if cur.right != null and cur.right.value not exist in explored:
        Cur = cur.right
    Else:
        Cur = cur.father
```

سوال امتیازی) فرض کنید دو تا لینکد لیست ساده داریم که طول اولی m ودومی n باشد و دقیقا نمیدانیم که m>n است یا برعکس. این دو تا لینکد لیست از یک نود خاص به بعد کاملا دارای نود های مشترک می شوند.

از شما می‌خواهیم الگوریتمی ارائه دهید که در اردر زمانی O(m+n) این نود خاص که اولین اشتراک این دو لینکد لیست هست را پیدا کند.

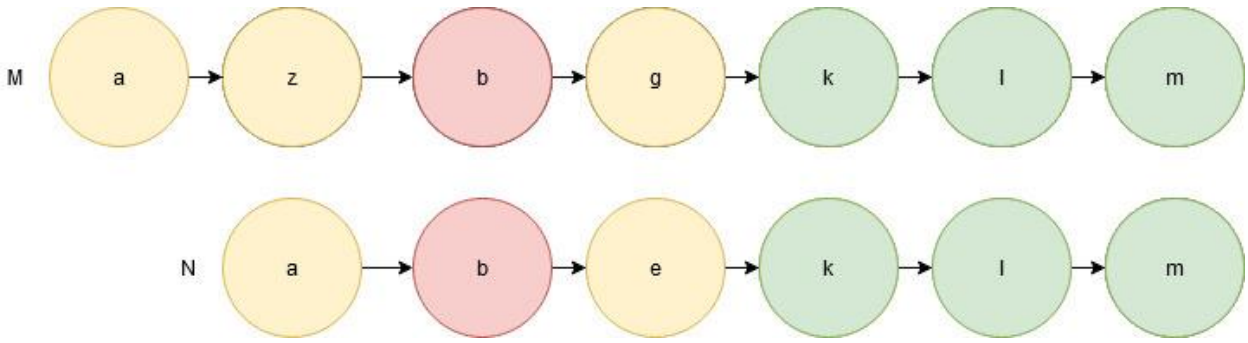
ابتدا یک بار از ابتدا تا انتهای هردو لینکد لیست میرویم و تعداد نود هایشان را درمیاوریم. این عملیات از O(m+n) می باشد.

حال لینکد لیستی که طول طولانی تری دارد را m و آنی که طول کوتاهتر دارد را n در نظر میگیریم(m>n). از نود اول m شروع میکنیم و به اندازه ی m-n نود جلو می آییم. از این نود می تواند اشتراک لینکدلیست های m,n شروع شود. لینکدلیست m را از همان نود و لینکدلیست n را از نود اول شروع به پیمایش میکنیم و آدرس نودهای m,n را متناظرا مقایسه میکنیم.

اولین جایی که آدرس هر دو نود یکی شده بود را در متغیر commonNode ذخیره میکنیم. سپس در پیمایش ادامه ی نود ها اگر مقدار هیچ دو نودی متفاوت نبود حواب ما همان commonNode است. ولی اگر در ادامه ی پیمایش جایی مقادیر متفاوت بود باید جلوتر برویم و دوباره وقتی به اشتراک رسیدیم commonNode را برابر نود جدید قرار میدهیم. الگوریتم ما زمانی به اتمام میرسد که به انتهای هر دو لینکدلیست رسیده باشیم.

برای مثال:

فرض کنید لیست های پیوندی M به طول ۷ و N به طول ۶ را مانند شکل زیر داریم. طبق الگوریتم بالا میبینیم طول M بیشتر است پس از خانه ی اول M، یعنی a شروع میکنیم و به اندازه ی m-n یعنی ۱ = ۶ - ۷ خانه جلو می آییم و به نود ای با محتوای b می رسیم. حال همزمان از همین خانه در لینکدلیست M و از خانه ی اول در لینکدلیست N شروع به پیمایش به صورت متناظر میکنیم. یعنی در هر مرحله آدرس نود های Mو N را مقایسه میکنیم و هم در M و هم در N یک نود به جلو میرویم و سپس آدرس ۲ نود جدید را مقایسه میکنیم و به همین منوال تا انتهای لیست ادامه میدهیم.



مراحل پیمایش هر دو لینکدلیست به طور متناظر:

مرحله۱:

لینکدلیست M: نود ای با مقدار z

لینکدلیست N: نود ای با مقدار a

آدرس نود ها یکی نیست

مرحله۲:

لینکدلیست M: نود ای با مقدار b

لینکدلیست N: نود ای با مقدار b

آدرس نود ها یکی است => commonNode را برابر نود فعلی میگذاریم.

مرحله۳:

لینکدلیست M: نود ای با مقدار g

لینکدلیست N: نود ای با مقدار e

آدرس نود ها یکی نیست => commonNode را null میکنیم.

مرحله۴:

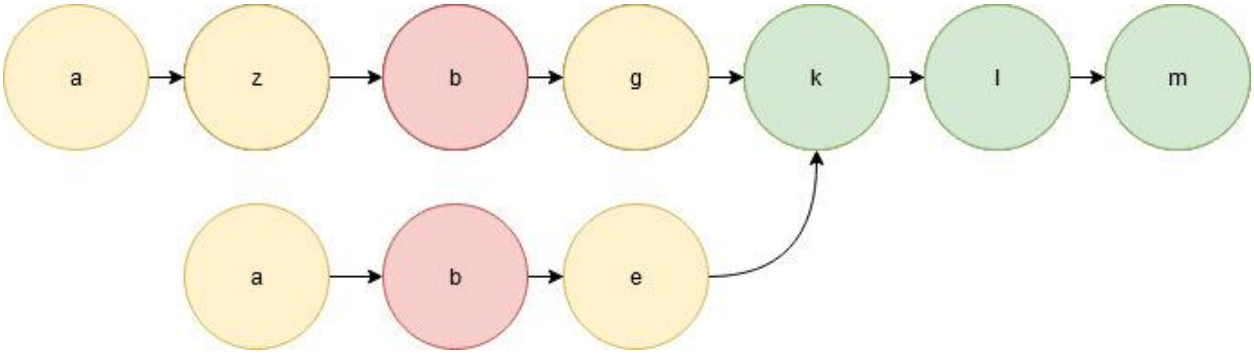
لینکدلیست M: نود ای با مقدار k

لینکدلیست N: نود ای با مقدارk

آدرس نود ها یکی است => commonNode را برابر نود فعلی میکنیم.

در مراحل بعدی تا انتهای لیست پیمایش میکنیم و میبینیم تمامی آدرس ها یکی است لذا commonNode را عوض نمیکنیم و مقدار commonNode همان مقداری است که در مرحله ی ۴ بدست آمد.

در واقع ۲ لینکدلیست ما چنین ساختاری داشته اند:



با استفاده از حافظه کمکی:

اول هر لینک لیست را در یک پشته میریزیم. سپس شروع به خارج کردن اعضا از پشته ها کرده و آنها را با هم مقایسه میکنیم هر جا متفاوت بودند از آنجا عضو های متفاوت داریم.

این سوال راه های دیگری نیز دارد.