

Compile

```
javac MusicPlayler.java
```

```
java MusicPlayer
```

```
shuqiny2@circinus-30 22:33:11 ~/253P/HW4
$ cd src
shuqiny2@circinus-30 22:33:24 ~/253P/HW4/src
$ javac MusicPlayer.java
shuqiny2@circinus-30 22:33:30 ~/253P/HW4/src
$ java MusicPlayer
```

Run

// red part: compile and run

// green part: output

```
shuginy2@circinus-30 22:33:24 ~/253P/HW4/src
$ javac MusicPlayer.java
shuginy2@circinus-30 22:33:30 ~/253P/HW4/src
$ java MusicPlayer
push Mundian To Bach Ke [Panjabi MC]
push My Immortal [Evanescence]
queue California Love [Tupac]
next
current
the current song is: California Love [Tupac]
the previous song is: Mundian To Bach Ke [Panjabi MC]
the next song is: My Immortal [Evanescence]
addBefore Mundian To Bach Ke [Panjabi MC] Canta Per Me [Yuki Kajiura]
addAfter Mundian To Bach Ke [Panjabi MC] Shape Of My Heart [Sting]
current
the current song is: California Love [Tupac]
the previous song is: Shape Of My Heart [Sting]
the next song is: My Immortal [Evanescence]
prev
prev
delete
changeTo California Love [Tupac]
current
the current song is: California Love [Tupac]
the previous song is: Shape Of My Heart [Sting]
the next song is: My Immortal [Evanescence]
find Mundian To Bach Ke [Panjabi MC]
cannot find song!
print
the playlist is:
0. My Immortal [Evanescence]
1. Canta Per Me [Yuki Kajiura]
2. Shape Of My Heart [Sting]
3. California Love [Tupac]
shuginy2@circinus-30 22:33:37 ~/253P/HW4/src
```

Q & A

1. What data structure did you implement **SimplePlaylist** as?

I implement SimplyPlaylist as a linked list, each song can be seen as a node.

2. List all the attributes (aka fields) you needed in order to implement **SimplePlaylist** (also include the attributes for any other auxiliary data structures it uses)? (Do not list functions (aka methods)).

```
Class SimplePlaylist {  
    protected Node head; // The first song of playlist  
    protected Node cur; // The current song of playlist  
    protected int size; // The size of playlist  
}  
  
class Node {  
    protected Song song;  
    protected Node next;  
}  
  
class Song {  
    protected String title;  
    protected String artist;  
}
```

3. How does **SimplePlaylist** retrieve a random song in $O(n)$ time? Explain in detail using a few sentences.

Get a random n (n is between 0 and the length of playlist), move the pointer from current position to its next n times.

4. If **prev** is processed in $O(n)$ time, then how is **find** able to print the previous song of the found song in $O(1)$ time?

The **cur** pointer can only move to the next node, so if you want to visit the previous node, you have to start from the beginning. That's how **prev** works in $O(n)$. But in **find**, since we have to traverse the whole list, we can keep the location of the previous song with an extra pointer. By using this pointer we can achieve $O(1)$ in **find**.

Two Leetcode problems are substitutions, but I'd like to briefly talk about some ideas about 4.2 and 4.3

5. Very briefly explain how **GeneralPlaylist** differs from **SimplePlaylist** as? (Just one or two short sentences that gets the main point(s) across).

SimplePlaylist is a linked list while **GeneralPlaylist** is a doubleLinkedList. Also, **GeneralPlaylist** keeps a pointer pointing to its tail.

6. List all the attributes (aka fields) you needed in order to implement **GeneralPlaylist** (also include the attributes for any other auxiliary data structures it uses)? (Do not list functions (aka methods)).

```
class DoubleLinkedListNode {
    protected Song song;
    protected DoubleLinkedListNode next;
    protected DoubleLinkedListNode pre;
    protected boolean visited;    // if this node has been visited
}

class GeneralPlaylist implements MusicPlayerImpl {

    protected DoubleLinkedListNode head; // The first song of playlist
    protected DoubleLinkedListNode tail; // The first song of playlist
    protected DoubleLinkedListNode cur;  // The current song of playlist
    protected int size; // The size of playlist
    static int randomCnt;

}
```

7. What allows queue to be able to add to the end of the playlist in $O(1)$ time now? We keep a tail pointer. Every time queue an element, just append to the tail and move the pointer.

8. How does **GeneralPlaylist** retrieve a random song without iterating over a song twice? Explain in detail using a few sentences.

Every node has a param `visited`, which will be initialized to false.

Add a param `randomCnt` to **GeneralPlaylist**, counting the number of songs that was

randomly retrieved.

When `random()` is called, we firstly do the same traversal in `SimplePlaylist`, then check if this node has been visited. If hasn't been visited, update `visited`, increment `randomCnt` and retrieve this song; else go next until an unvisited node is found.

When `randomCnt` equals to the size of the playlist, reset `randomCnt` to 0 and every node to unvisited.

9. Very briefly explain how ***AdvancedPlaylist*** differs from ***GeneralPlaylist*** as? (Just one or two short sentences that gets the main point(s) across).

AdvancedPlaylist has one more `HashMap<String, DoubleLinkedNode>`, whose key is the title and artist of a song and the value is the node in playlist.

10. What additional data structure did you use to help you achieve the desired improvements?

A `hashMap`.

Leetcode 1019

The screenshot shows the LeetCode interface for the "Next Greater Node In Linked List" problem. The user has successfully submitted a Java solution. The runtime is 14 ms, faster than 78.90% of other submissions. The memory usage is 44.1 MB, less than 6.01% of other submissions. Below the performance metrics, there are three challenge buttons: "Basic Calculator", "Remove K Digits", and "Make The String Great". A table displays the submission history with columns for Time Submitted, Status, Runtime, Memory, and Language.

Time Submitted	Status	Runtime	Memory	Language
11/03/2020 22:50	Accepted	14 ms	44.1 MB	java
10/28/2020 16:10	Accepted	14 ms	43.4 MB	java
10/28/2020 11:43	Wrong Answer	N/A	N/A	java

```

Stack<Integer> stack = new Stack();
int[] res = new int[n];
p = head;
int cnt = 0;

while(p != null) {
    if(stack.isEmpty()) {
        stack.push(p.val);
        res[cnt] = 0;
    }
    else {
        if (p.val < stack.peek()) {
            res[cnt] = 0;
            stack.push(p.val);
        }
        else {
            int i = cnt - 1;
            while(!stack.isEmpty() && stack.peek() < p.val) {
                if (res[i] == 0) {
                    res[i] = p.val;
                    stack.pop();
                }
                i--;
            }
            res[cnt] = 0;
            stack.push(p.val);
        }
    }
    p = p.next;
    cnt++;
}
return res;
}

```

Leetcode 1035

LeetCode

Explore

Problems

Mock

Contest

Discuss

Store

November LeetCode Challenge

Premium

力和「中文社区」现已上线，全新「个人主页」更优体验，即刻加入先人一步攒积分！

新功能推荐：

全新积分商场礼品

 | 上万社区题解 | 企业题库 | 面试模拟 | 更多竞赛 | 轻松数据同步使用已有积分和会员

Description

Solution

Discuss (360)

Submissions

Success

Details

Runtime: 3 ms, faster than 99.83% of Java online submissions for Uncrossed Lines.

Memory Usage: 37.1 MB, less than 9.42% of Java online submissions for Uncrossed Lines.

Next challenges:

Edit Distance

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
11/04/2020 00:11	Accepted	3 ms	37.1 MB	java
10/31/2020 14:10	Accepted	3 ms	36.8 MB	java
10/31/2020 14:06	Accepted	3 ms	38.6 MB	java

Java

Autocomplete

```
1 class Solution {
2     public int maxUncrossedLines(int[] A, int[] B) {
3         int m = A.length, n = B.length;
4         if (m < n) return maxUncrossedLines(B, A);
5
6         int dp[] = new int[n + 1];
7         for (int i = 1; i <= m; i++) {
8             int prev = 0;
9             for (int j = 1; cur; j <= n; j++) {
10                 cur = dp[j];
11                 if (A[i - 1] == B[j - 1])
12                     dp[j] = 1 + prev;
13                 else
14                     dp[j] = Math.max(dp[j], dp[j - 1]);
15                 prev = cur;
16             }
17         }
18         return dp[n];
19     }
20 }
```