Warm-up

Problem 1. Suppose we implement a stack using a singly linked list. What would be the complexity of the push and pop operations? Try to be as efficient as possible.

Problem 2. Suppose we implement a queue using a singly linked list. What would be the complexity of the enqueue and dequeue operations? Try to be as efficient as possible.

Problem solving

Problem 3. We want to extend the queue that we saw during the lectures with an operation getAverage() that returns the average value of all elements stored in the queue. This operation should run in $O(1)$ time and the running time of the other queue operations should remain the same as those of a regular queue.

a) Design the getAverage() operation. Also describe any changes you make to the other operations, if any.

b) Briefly argue the correctness of your operation(s).

c) Analyse the running time of your operation(s).

Problem 4. Given a singly linked list, we would like to traverse the elements of the list in reverse order.

a) Design an algorithm that uses $O(1)$ extra space. What is the best time complexity you can get?

b) Design an algorithm that uses $O(\sqrt{n})$ extra space. What is the best time complexity you can get?

You are not allowed to modify the list, but you are allowed to use position/cursors to move around the list.

Problem 5. Using only two stacks, provide an implementation of a queue. Analyze the time complexity of enqueue and dequeue operations.

Problem 6. Consider the problem of given an integer n, generating all possible permutations of the numbers $\{1, 2, \ldots, n\}$. Provide a recursive algorithm for this problem.

Problem 7. Consider the problem of given an integer n, generating all possible permutations of the numbers $\{1, 2, \ldots, n\}$. Provide a non-recursive algorithm for this problem using a stack.

Problem 3. We want to extend the queue that we saw during the lectures with an operation GETAVERAGE() that returns the average value of all elements stored in the queue. This operation should run in $O(1)$ time and the running time of the other queue operations should remain the same as those of a regular queue.

a) Design the GETAVERAGE() operation. Also describe any changes you make to the other operations, if any.

b) Briefly argue the correctness of your operation(s).

c) Analyse the running time of your operation(s).

(a) $\frac{\text{total()}}{\text{size()}}$

(b)

(c)

Queue: FIFO

. We should creat a new method, which functionality is:

① After enque, it adds the first()

② when deque: it minus the element of deque()

I prefer to call this function "total()"

. since both total() and size() just return value which complexity is O(1), therefore $\frac{\text{total()}}{\text{size()}}$ is O(1)

Problem 1. Suppose we implement a stack using a singly linked list. What would be the complexity of the push and pop operations? Try to be as efficient as possible.

- Pop and push are stack's function
- They simply just manipulate the 1st element
- For Push(), we can
  1. create a new node x with element e
  2. x.next connect to original node
  3. change head pointer to node x
- As for pop(), we can
  1. change the head pointer to x.next, since there is no pointer connect to x, x will be recycled
- Both operations have O(1) complexity in the worst case

1st Deq

Problem 2. Suppose we implement a queue using a singly linked list. What would be the complexity of the enqueue and dequeue operations? Try to be as efficient as possible.

- For enqueue():

✗ ① similar to push, so the complexity is O(1)

- For dequeue():

✗ since we need to traverse n-1 elements to find the predecessor of last node (due to singly linked node), therefore the complexity is O(n)

FI FO

For enque:
① we firstly record the address of last node
O(1) ② create new node
③ last.next to new node (Connect) ④ record new last node address

For deque ① just remove the 1st node
O(1) since it's the 1st element be enqued

O(1) extra space refers to an algorithm that requires a constant amount of additional memory space, regardless of the size of the input data.

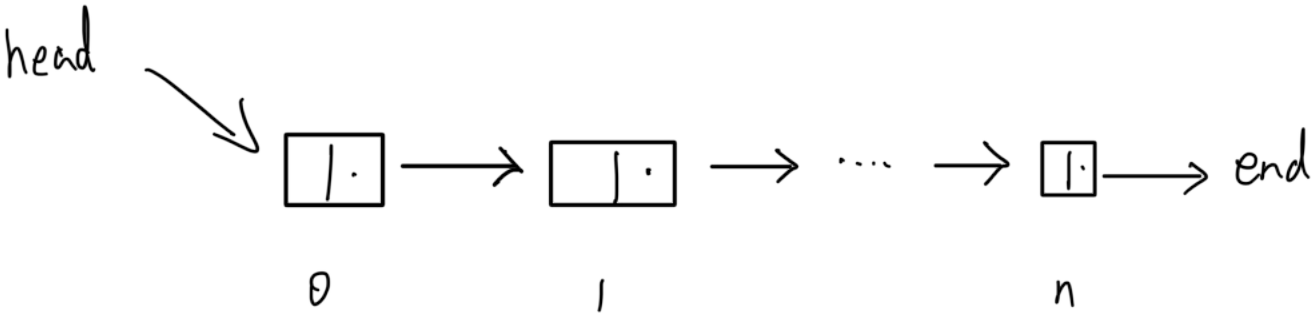
Problem 4. Given a singly linked list, we would like to traverse the elements of the list in reverse order.

a) Design an algorithm that uses $O(1)$ extra space. What is the best time complexity you can get?

b) Design an algorithm that uses $O(\sqrt{n})$ extra space. What is the best time complexity you can get?

You are not allowed to modify the list, but you are allowed to use position/cursors to move around the list.

O(1) extra space mean?



. without, the complexity is $O(1+2+\cdots+n = \frac{n(n+1)}{2})$

$= O(n^2)$

(or)

```
def reverse( ):
```

key idea is: we don't have former node address

How about we create a new linked list? no, since the space is O(n)

① store the middle node address ?

check answer

(b) 1 2 3 4 5 6 7 8 9 .....

3 6 9

$\sqrt{n}$ $2\sqrt{n}$ $3\sqrt{n}$

如果可以用 stack 呢

① 找到 list[$\sqrt{n}$], ..., list[$2\sqrt{n}$]

② 用 stack 存这些辅助的 list

③ 最后按顺序每个 pop 每个 list

Problem 5. Using only two stacks, provide an implementation of a queue. Analyze the time complexity of enqueue and dequeue operations.

Stack: FILO

Queue: FIFO

① store all enque elements in stack1

② whenever enque action stop, we begin to pop all elements in stack1, and store those element in stack2, then we do deque action for stack2

③ whenever we want to do any enque again, we need to move all elements in stack2 to stack1 before the enque action

En/push	De	pop
5		
4	5	4
3	4	5
2	3	1
1	2	
	1	3

- 更好是在如果 S_2 is empty
  我把 S_1 移动到 S_2 (POP的方式)
- 所有 push 都往 S_1
- pop 时 S_2 有值则 pop S_2
  否则先依次将 S_1 到 S_2
  再 pop S_2

Permutation：排列

n!

Problem 6. Consider the problem of given an integer n, generating all possible permutations of the numbers $\{1, 2, \ldots, n\}$. Provide a recursive algorithm for this problem.

return a list

```
def per(list):
    if list = 0:
        return[null]
    if list = 1:
        return[list[0]]
    else:
        for i in list:
            res.append(i + per(list))
        return res
```

有问题，因为 list 应该包含 i

↳ 刚刚这里应该用 for 表示

正确答案：

```
def permutation(lst):  #lst = [0,1,2...,n]
    if len(lst) == 1:
        return [lst]
    else:
        res = []
        for i in lst:
            exclude_i_list = [x for x in lst if x != i]
            for p in permutation(exclude_i_list)
                res.append([i] + p)
        return res
```

#不包含 i 的 lst

这里 P 表示不包含 i 的所有 Permutation，在前 P 前面加上 i 即表示

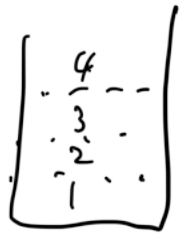
$n \cdot (n-1) \cdots (1) = n!$

Problem 7. Consider the problem of given an integer n, generating all possible permutations of the numbers $\{1, 2, \ldots, n\}$. Provide a non-recursive algorithm for this problem using a stack.

X

X is the list

stack: FILO



just do $1 \times \cdots \times 3 \times 2 \times 1 = n!$

```
res = 1
for i in stack:
    res = res x i
```