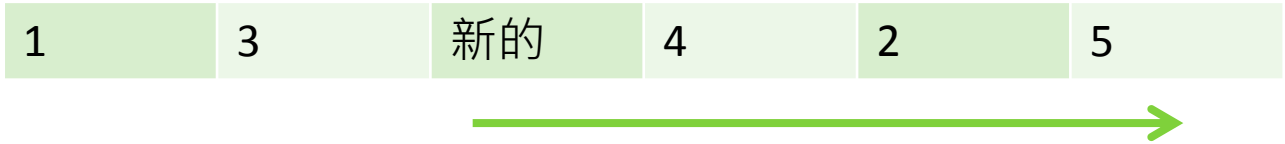
LINKED LISTS

Prof. Michael Tsai

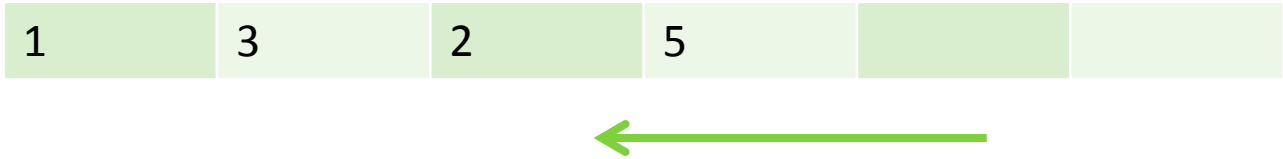
2013/3/12

大家來吐Array的槽

- Array有什麼不好?
- 插入新element



- 刪除原本的element



- Time complexity= O(??)

Array的複雜度

	Array	Dynamic Array (滿了以後 擴充成兩倍)	Linked List (今天要學的)
Indexing (拿某一個元素)			??
在開頭 Insert/Delete			??
在尾巴 Insert/Delete			??
在中間 Insert/Delete			??
浪費的空間			??

新朋友: Linked List

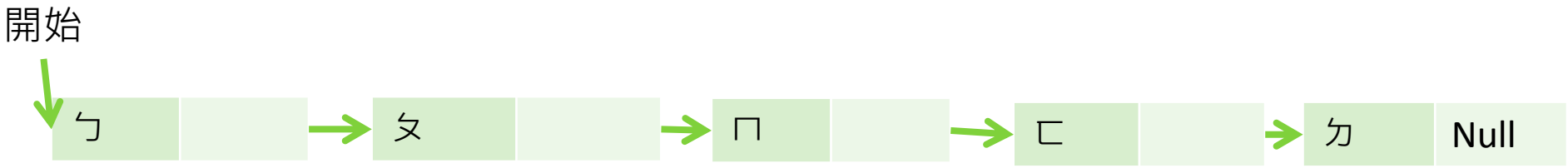
- 要怎麼讓資料
- 1. 可以隨便亂排
- 2. 但是我們仍然知道他的順序?

- 答案:
- 資料亂排
- 但是, 另外存“下一個是誰”

index	[0]	[1]	[2]	[3]	[4]
資料	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ
下一個是誰	4	0	1	-1	3

開始	1

概念上, 應該是長這樣

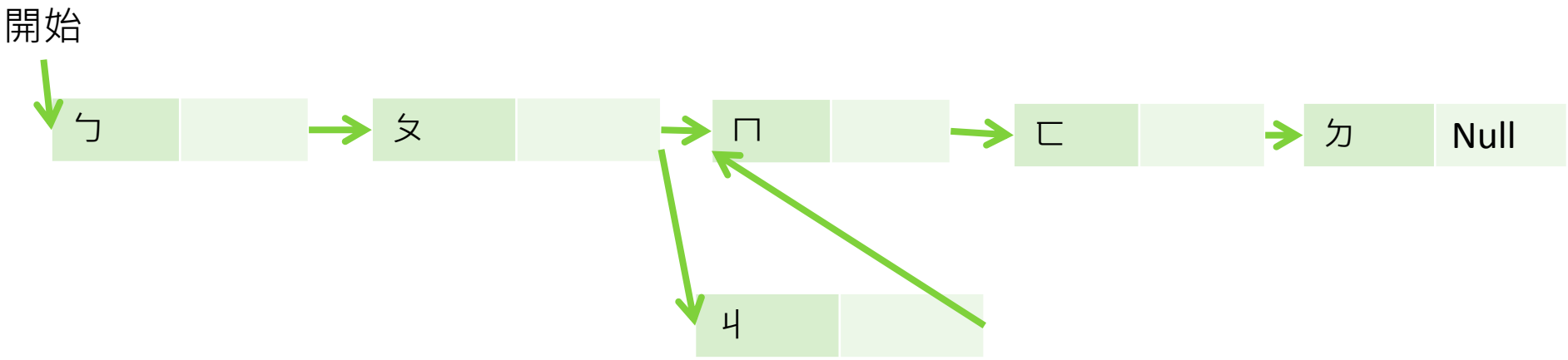


真正的樣子:

index	[0]	[1]	[2]	[3]	[4]
資料	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ
下一個是誰	4	0	1	-1	3

開始 2

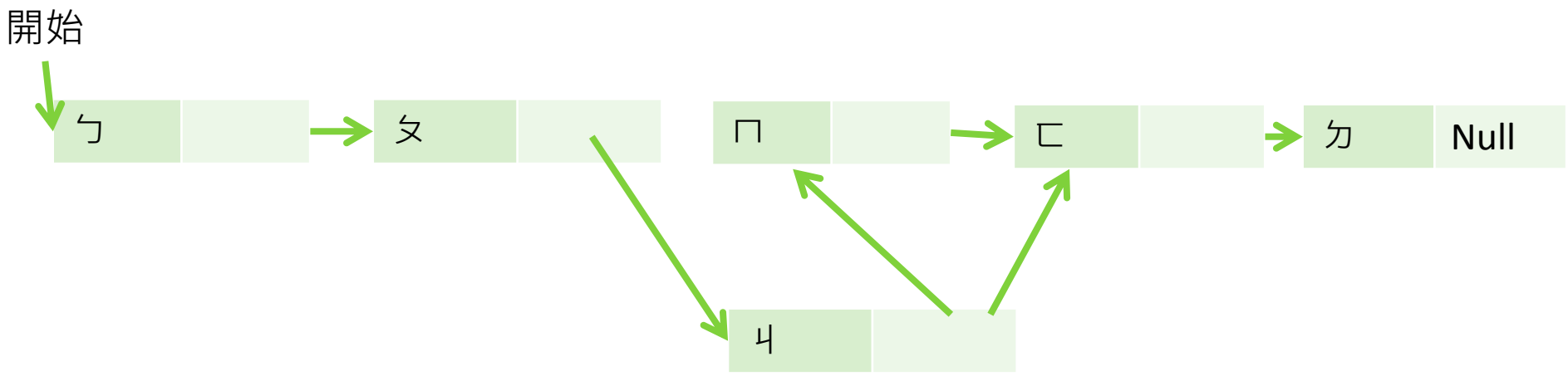
增加一個人?



index	[0]	[1]	[2]	[3]	[4]	[5]
資料	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ	ㄐ
下一個是誰	4	5	1	-1	3	0

開始	2

刪掉一個人



index	[0]	[1]	[2]	[3]	[4]	[5]
資料	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ	ㄐ
下一個是誰	4	5	1	-1	3	4

開始	2

來看一些code (用malloc/free)

- 怎麼宣告一個node的struct?

```
struct ListNode {
  int data;
  struct ListNode *next;
};
```

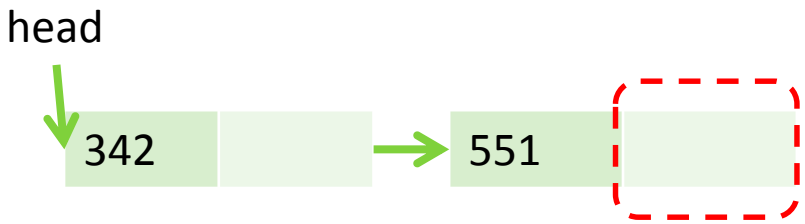
- 怎麼拿一個新的node?
- `struct ListNode *new;`
- `new=(struct ListNode*)malloc(sizeof(struct listNode));`

來看一些code

- new是指標, 指到一個struct listNode的變數.
- 那如果要拿這個變數裡面的data呢?
- 可以這樣寫:
- (*new).data
- 或者,
- new->data

- 要拿next呢?
- (*new).next
- 或者,
- new->next

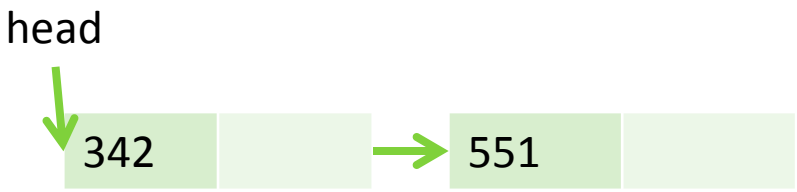
來看一些code



- 假設head指到第一個struct ListNode

- 那麼我要拿到551的下一個ListNode的位址怎麼寫?
- head->link->link (連續技)

製造兩個node



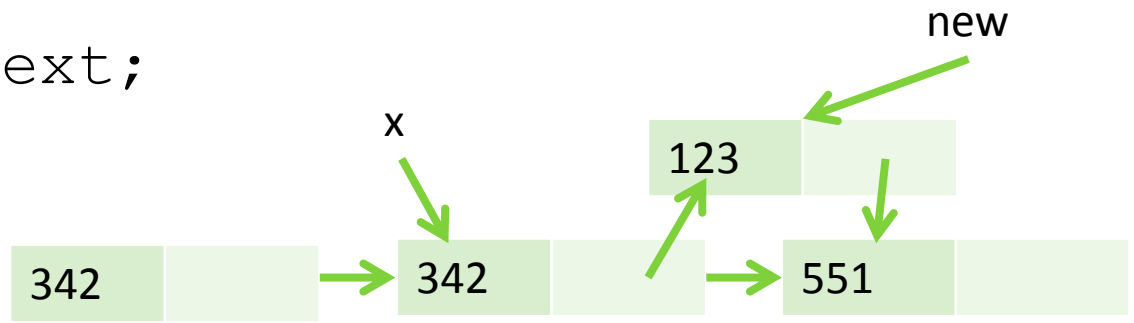

```
struct ListNode *head, *tmp;

tmp=(struct ListNode*)malloc(sizeof(struct ListNode));
if (tmp==NULL)
      exit(-1); // exit program on error
tmp->data=551;
tmp->next=NULL;
head=tmp;
tmp=(struct ListNode*)malloc(sizeof(struct ListNode));
tmp->data=342;
tmp->next=head;
head=tmp;
```

插入一個新node在某node後面

- `struct ListNode *x; //指到要插入的node的位置`
- `struct ListNode *new;`

- `new=(struct ListNode*)malloc(sizeof(struct ListNode));`
- `new->data=123;`

- 下一步呢？先處理`new->next`還是`x->next`？
- `new->next=x->next;`
- `x->next=new;`

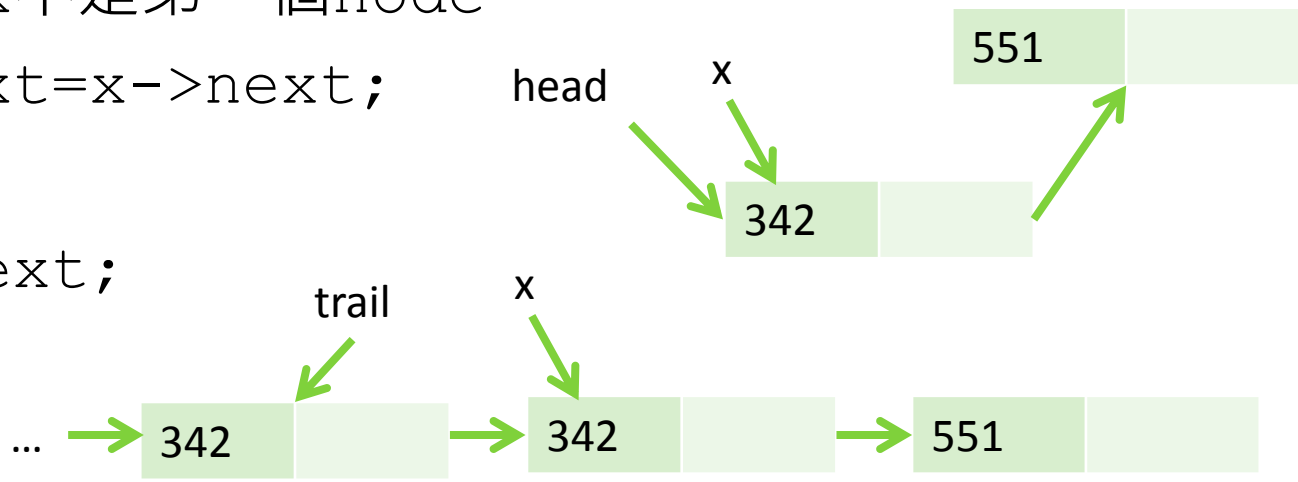


刪除某一個node

- `struct ListNode *head;` //指到一開始的node的位置
- `struct ListNode *x;` //指到要刪除的node的位置
- `struct ListNode *trail;` //指到x的前一個node的位置

- 分兩種狀況處理: x是頭, 還有x不是頭

```
if (trail)   //x不是第一個node
      trail->next=x->next;
else
      head=x->next;
free(x);
```



Examples

- 印出整個linked list

```
struct ListNode *tmp;
for(tmp=head; tmp!=NULL; tmp=tmp->next)
      printf("%d ", tmp->data);
```

- 找到某個node有data的前一個node的位置

```
int a=123; //假設我們要找資料是123的node的前一個node位置
struct ListNode *tmp;
for(tmp=head; tmp!=NULL; tmp=tmp->next) {
      if (tmp->next!=NULL) { //因為tmp->next可能是NULL!
            if (tmp->next->data==a)
                  break; // break出去的時候, tmp就是我們要的
      }
}
```

Array和Linked List的複雜度比較

	Array	Dynamic Array (滿了以後 擴充成兩倍)	Linked List
Indexing (拿某一個元素)	O(1)	O(1)	
在開頭 Insert/Delete	O(n), only feasible if not full	O(n)	
在尾巴 Insert/Delete	O(1), only feasible if not full	O(1), if not full O(n), if full	
在中間 Insert/Delete	O(n), only feasible if not full	O(n)	
浪費的空間 (不是拿來存資料的部分)	0 (存滿以後)	O(n) (最多可能有一半的空間是空的)	

<動腦時間>

- 有沒有什麼是array比linked list好的?
- 什麼時候用array?
- 什麼時候用linked list?
- <練習題1> 我想要找linked list裡面從尾巴數過來的第k個node, 要怎麼寫code? 時間複雜度為?

練習題1答案: O(n), Karumanchi 3.10 problem 2,4,5

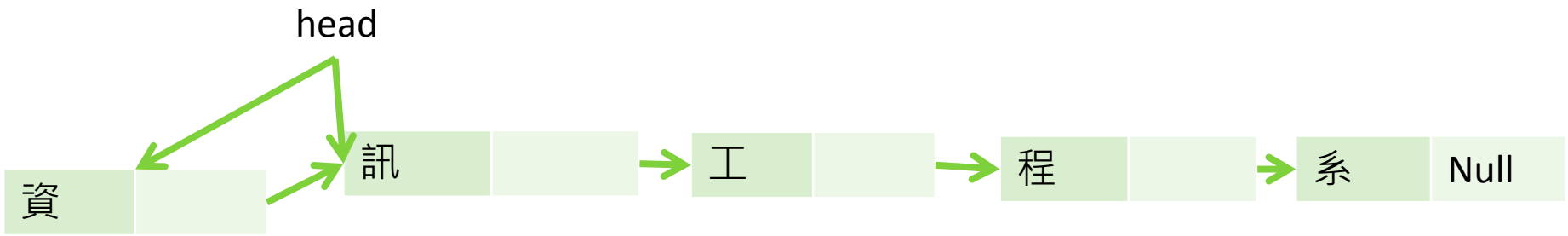
Example: Stacks & Queues

- 如果是一塊記憶體要放很多stack或queue
- 就很難做到很efficient
- 例如如果某一stack滿了, 就要把一堆資料往後擠
- 就不是O(1)了 T_T

- 解決: 跟Linked List當朋友

Stack

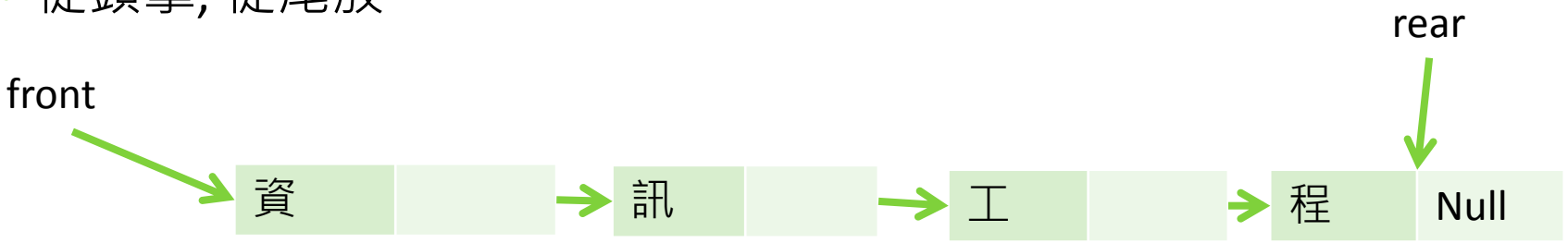
- 要怎麼拿來當stack呢? (想想怎麼做主要的operation)
- push & pop
- 請一位同學來講解☺



- 例: push(“學”)
- head當作stack top
- 怎麼寫code?
- 那pop呢?

Queue

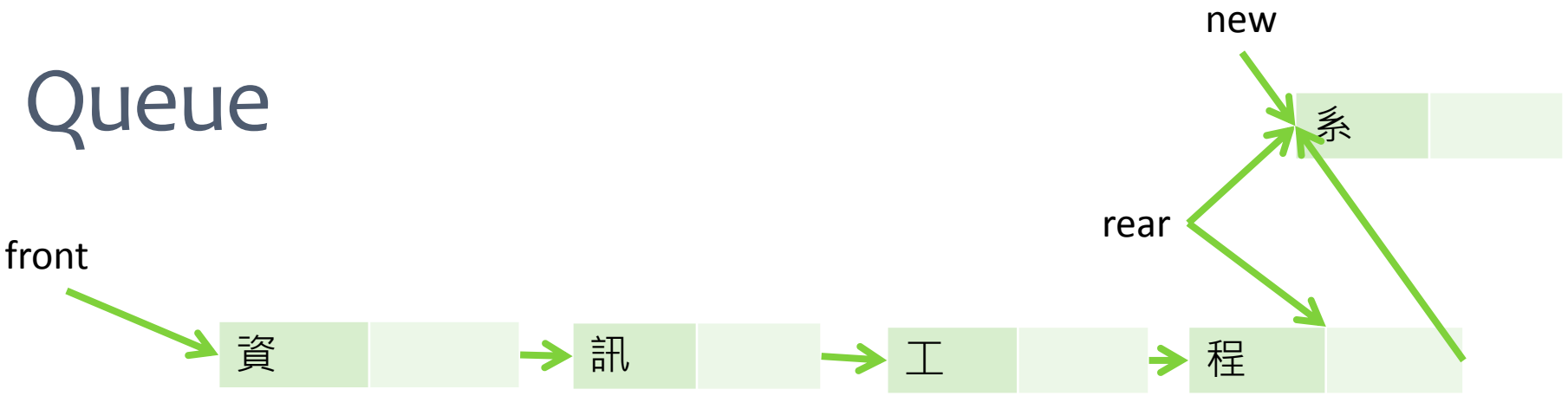
- 類似stack的作法
- 不過頭尾都要有一個指標
- 從頭拿, 從尾放



- 怎麼拿? (DeQueue)

```
struct ListNode* tmp;
tmp=front;
front=front->link;
tmp_data=tmp->data;
free(tmp);
return tmp_data;
```

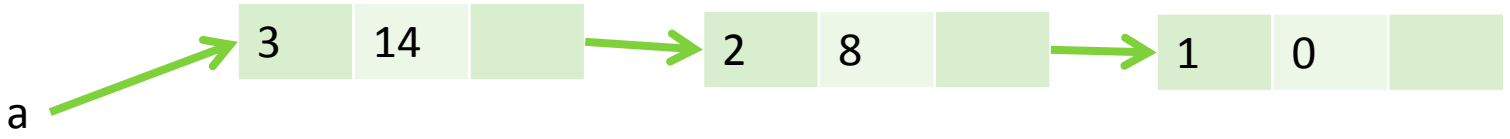
Queue



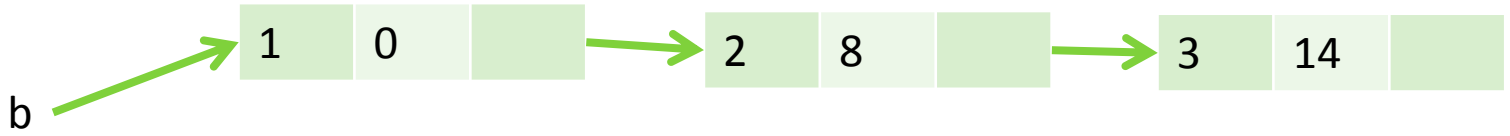
- 那怎麼放?
- 假設new是指到新的node
- rear->next=new;
- new->next=NULL;
- rear=new;

練習題2: 把linked list反過來

- 反過來: 把



- 變成

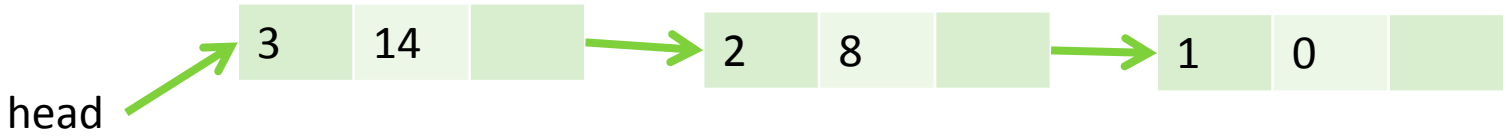


- 怎麼弄?

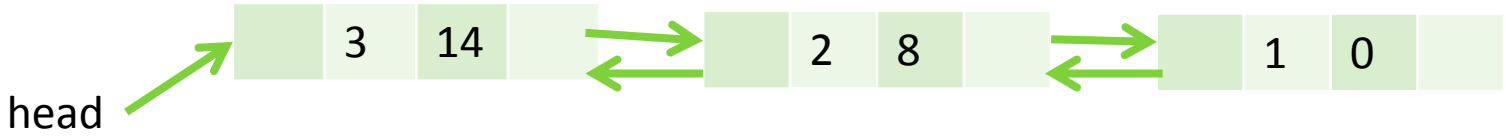
答案: Karumanchi 3.10 problem 16

Singly v.s. doubly linked list

Singly linked list:



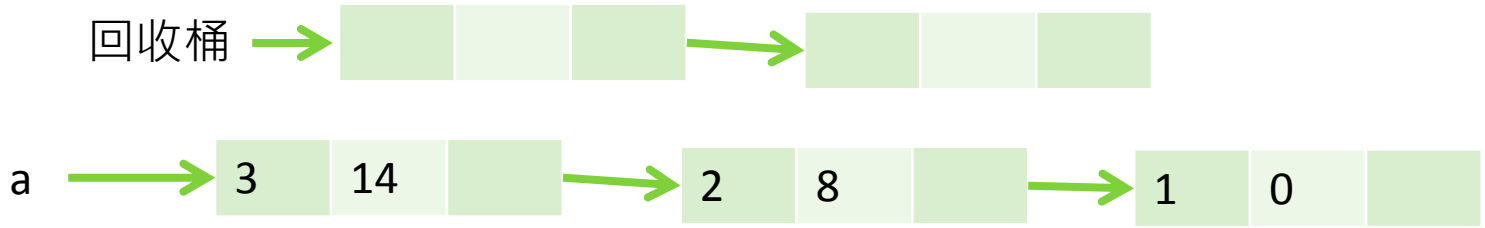
Doubly linked list:



- 什麼時候需要用雙?
- Singly linked list只能往後, 不能往前 (要從最前面開始重新找)
- Doubly linked list用在常常需要"倒帶"的時候
- 好處:
  - 倒帶方便 (O(1))
    (想想在singly linked list裡面要刪掉一個node時, 必須要找到前一個node)
- 壞處:
  - 兩個pointer的儲存空間
  - Insert/delete的時候稍微慢一點(要處理兩個pointer, next & prev)

回收很慢

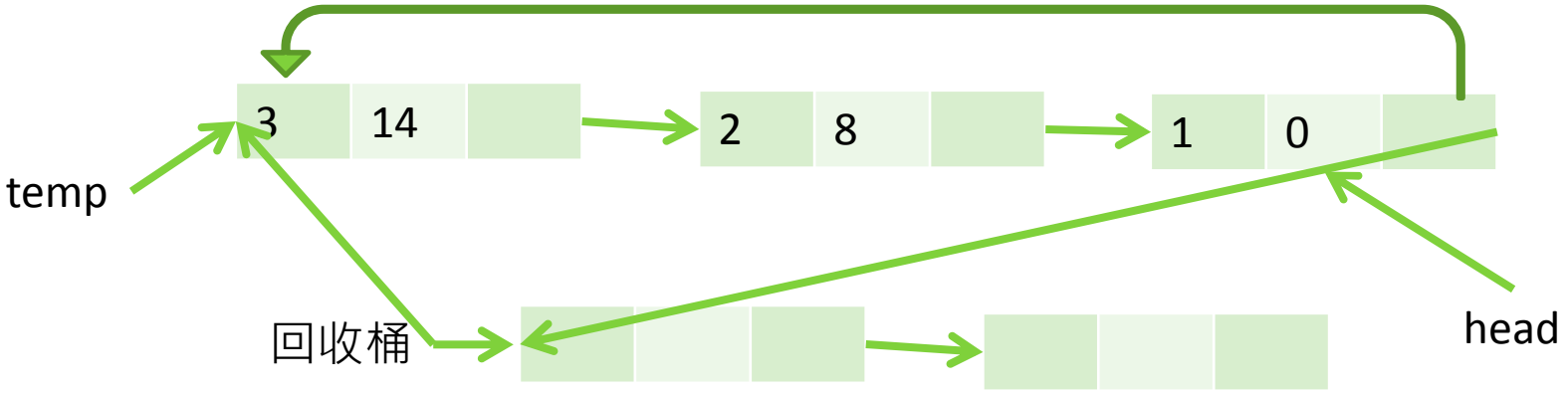
- 要把一個用完的linked list每個node都free掉 (Why?)
- 有幾項就要幾項的時間: O(n)
- 懶人方法: 丟到一個"回收桶", 之後需要的時候再撿出來
- 希望丟=O(1), 而且撿=O(1)
- 怎麼做?



- 關鍵:找尾巴很慢. (不是O(1))
- 但是又不想多花空間紀錄尾巴

Circular List

- “開始”的那個箭頭, 指在尾巴
- 最後一個node指回開頭
- 有什麼好處? 串接很方便!
- 丟進回收桶=O(1) !!



- 也可以有doubly circular linked list!

暫停!

- 來回想一下我們學了哪些種類的linked list

- Singly linked list
  - Circular
  - Non-circular (chain)
- Doubly linked list
  - Circular
  - Non-circular (chain)

Today's Reading Assignments

- Karumanchi 3.1-3.8
  - 有很多的source code, 特别是3.6的部分是基礎, 一定要看懂!
- Karumanchi 3.10 problem {2,4,5},{15,16},{24,25,27}