cpshCP 03. STL/basic DS

3-1. queue/stack/linklist

什麼是資料結構(data structrue, DS)

資料結構是資料在電腦中儲存、組織的方式，

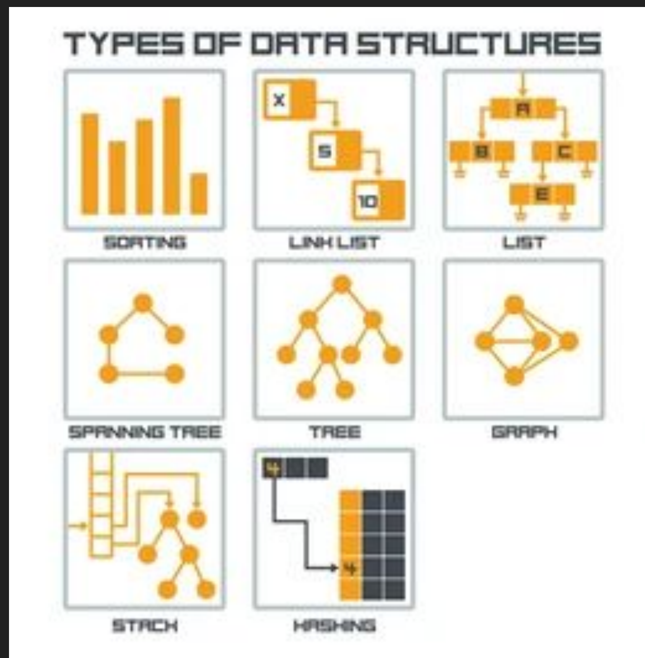
不同的資料結構會有各自擅長的應用，

像是array可以隨機存取(random access)、

linklist可以O(1)插入/刪除、

紅黑樹可以O(lgSize)尋找樹上是否有某個元素等等...

聰明的選擇使用的資料結構可以大大提升程式的效率！



stack(堆疊)

stack就像是吃迴轉壽司的時候,

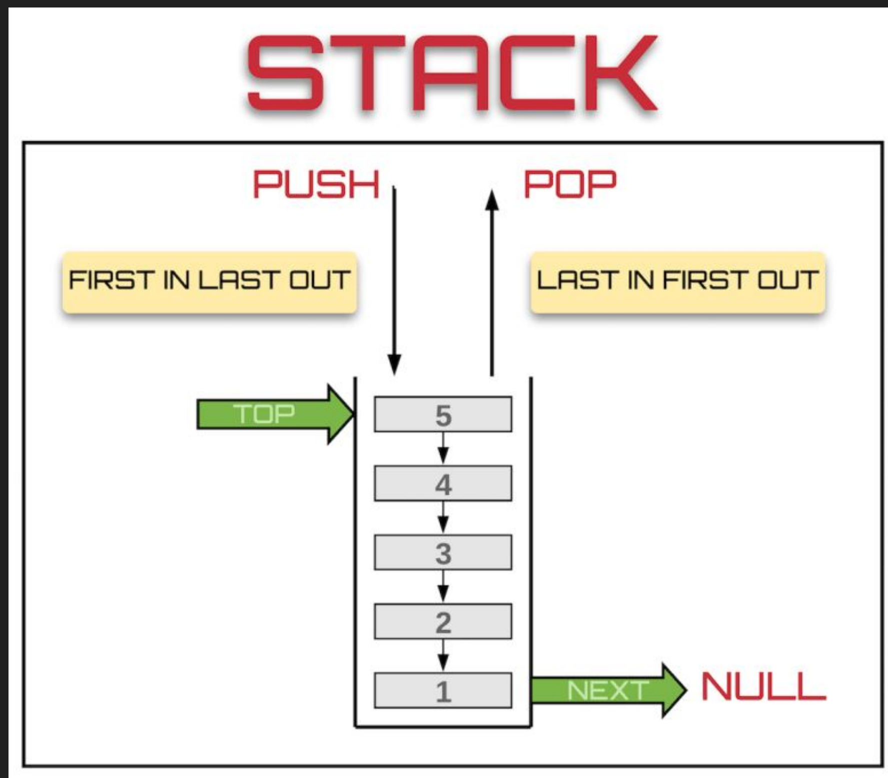
堆起來的盤子，可以把盤子放到最上面,

又或是把最上面的盤子拿走。

像這樣能**插入/刪除頂端元素**,

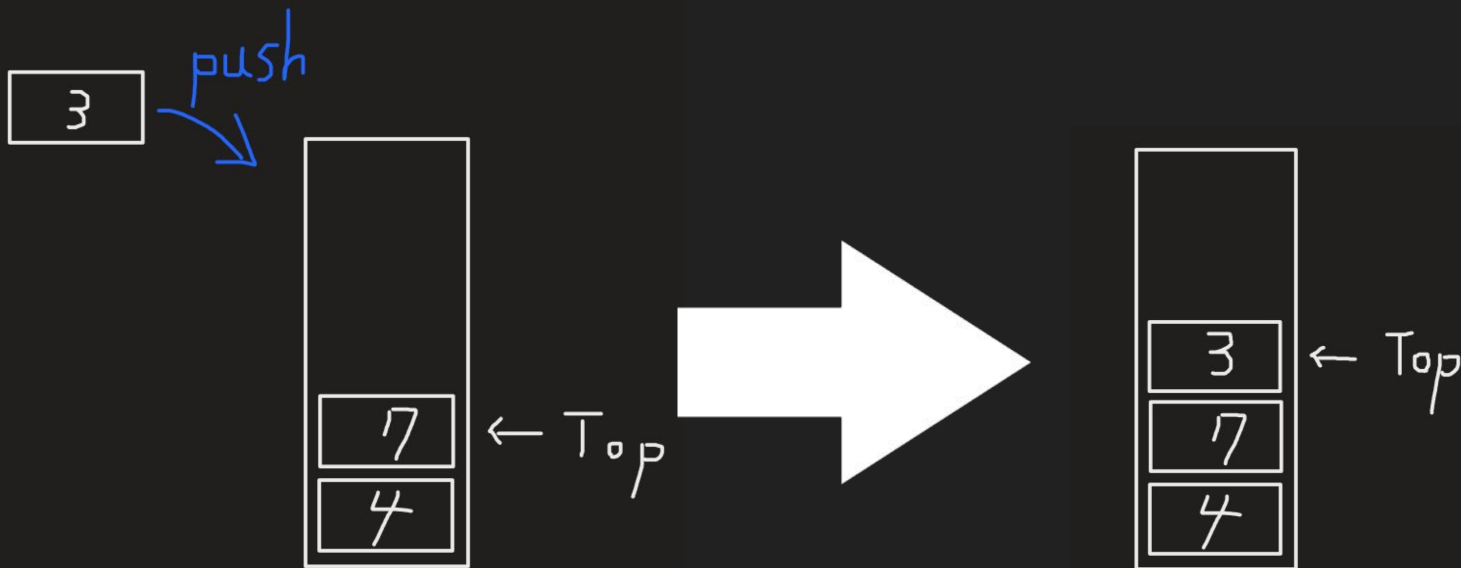
是一個FILO(先進後出)的資料結構。

ps. 不考慮吃壽司時把盤子插到中間的人。



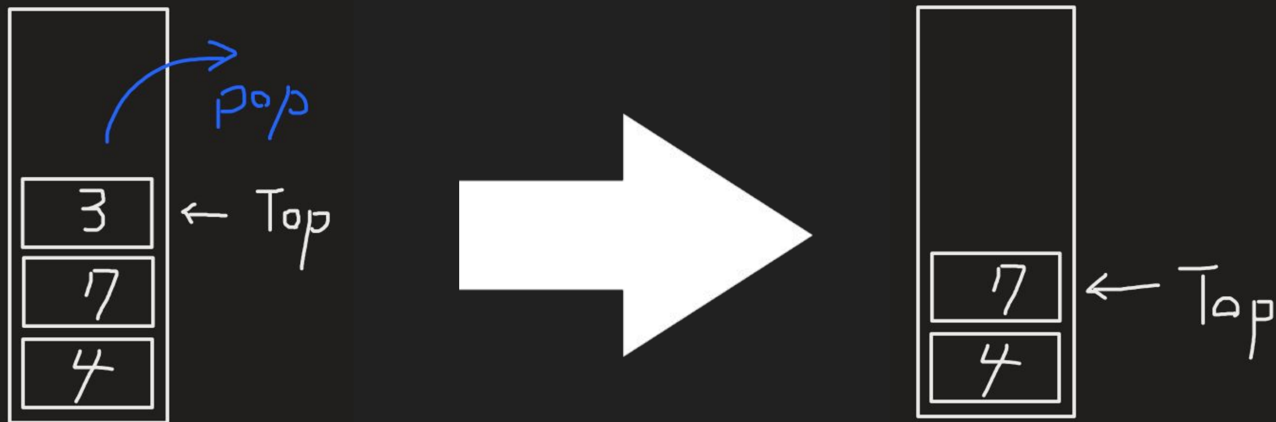
stack支援的操作-插入(push)

把一個元素放到stack最頂端。



stack支援的操作-刪除(pop)

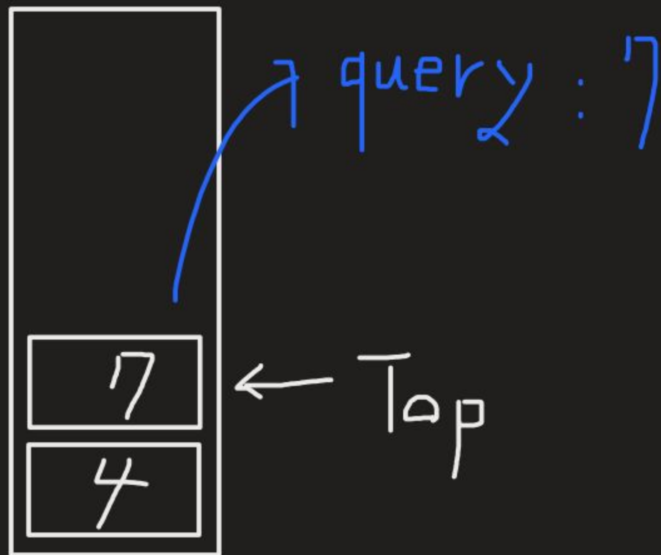
把stack頂端的元素移除。



stack支援的操作-查詢(query)

存取stack頂端的元素。

(stack不能隨機存取，所以你只能知道最頂端的元素是什麼)



stack實作

1. 靜態陣列
2. linked list

兩種的push/pop/query都是O(1)！

這裏只說明靜態陣列的實作方式,

linked list的實作方式等到學完之後可以想想看怎麼做～

stack實作

簡單來說就是開一個變數記錄stack

頂端的元素位置，剩下的東西就水到渠成了。

```
struct stack {
  static const int SIZE = 10000;
  int arr[SIZE];
  int cur = -1;

  void insert(int val) {
    if (cur == SIZE - 1)
      return;
    arr[++cur] = val;
  }
  void pop() {
    if (cur == -1)
      return;
    cur--;
  }
  int top() {
    if (cur == -1)
      return -1;
    return arr[cur];
  }
};
```

stack常見應用

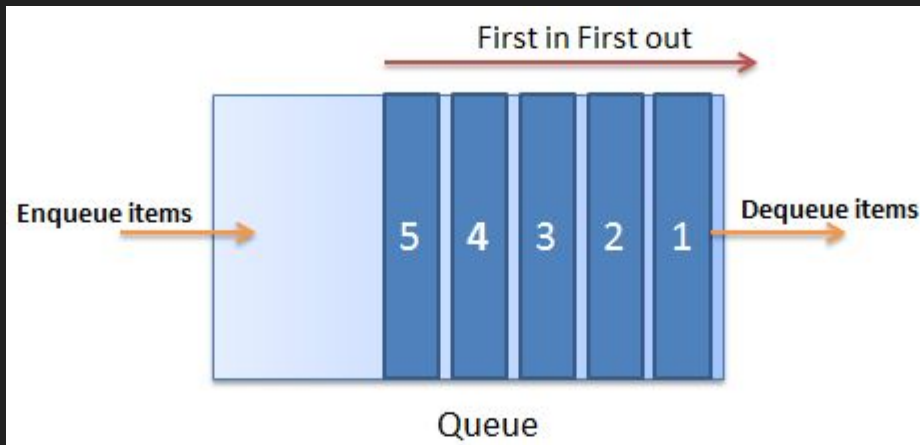
1. 括號配對(判定一串括號是不是合法的問題, ex. (()()) -> 合法,)()(-> 不合法)
2. 後序運算式
3. 模擬遞迴

queue(隊列)

隊列就和平常在排隊一樣,

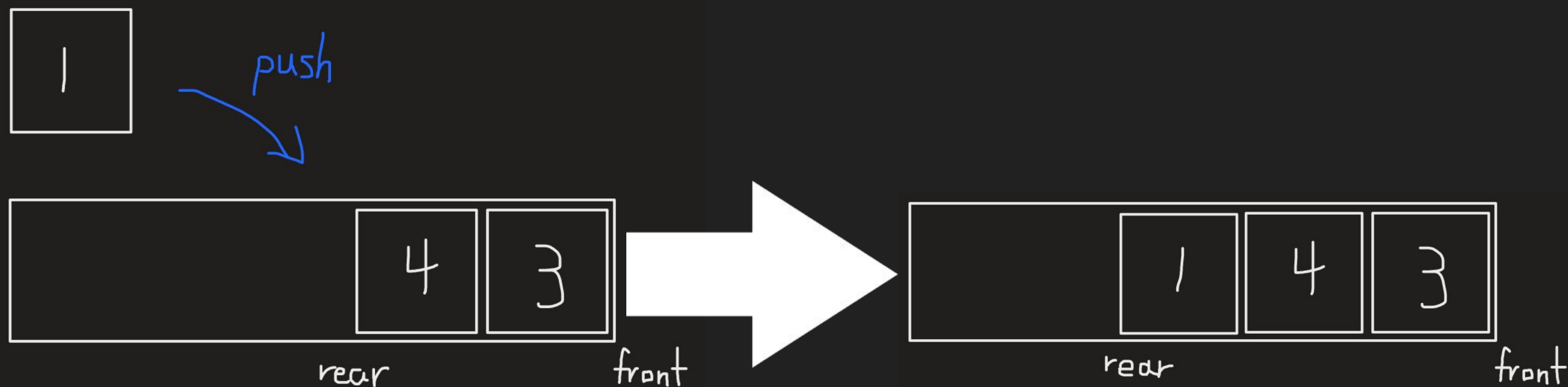
元素可從後方加入,從前方離開。

是一個FIFO(先進先出)的資料結構。



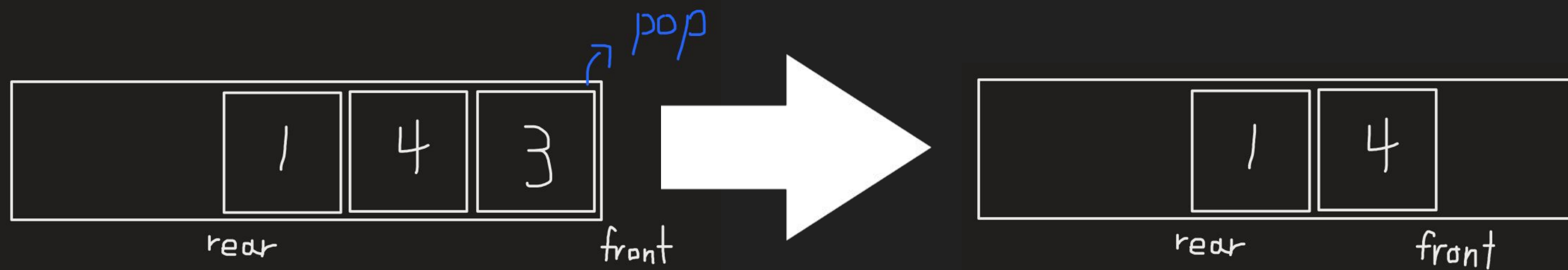
queue支援的操作-插入(push)

插入一個元素到隊列最尾端。



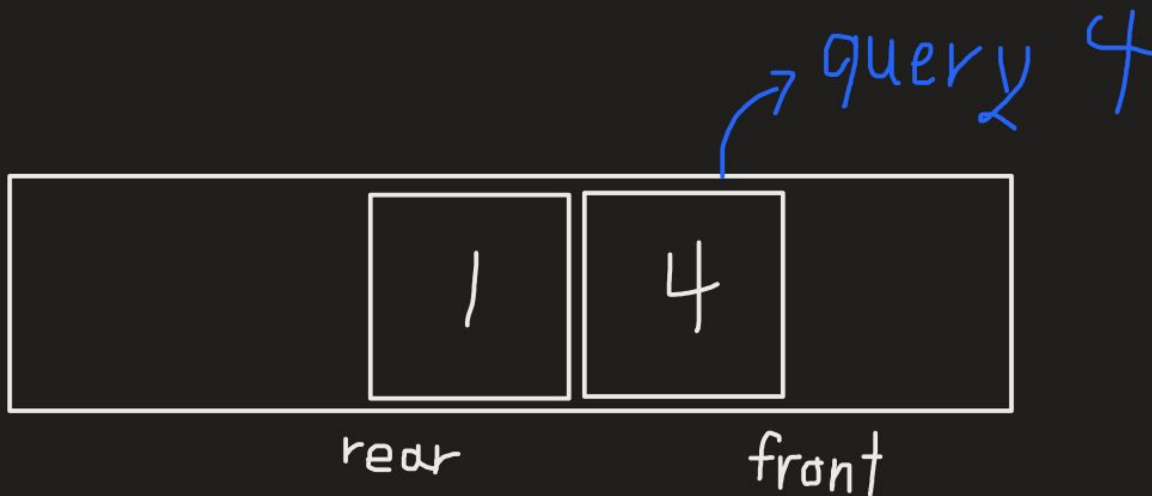
queue支援的操作-刪除(pop)

從隊列最前端刪除一個元素。



queue支援的操作-查詢(query)

詢問隊列最前端的元素。



queue實作

1. 靜態陣列
2. linked list

一樣兩種實作的方式push/pop/query都是O(1)。

queue實作

跟stack很像，只是這次同時維護了

queue最前端跟最後端的位置。

```
struct queue {
  static const int SIZE = 10000;
  int arr[SIZE];
  int front = 0;
  int rear = -1;
  void push(int val) {
    rear++;
    arr[rear % SIZE] = val;
  }
  void pop() {
    if (rear < front)
      return;
    front++;
  }
  int query() {
    if (rear < front)
      return -1;
    return arr[front % SIZE];
  }
};
```

queue實作

為什麼要取模?

```
struct queue {
  static const int SIZE = 10000;
  int arr[SIZE];
  int front = 0;
  int rear = -1;
  void push(int val) {
    rear++;
    arr[rear % SIZE] = val;
  }
  void pop() {
    if (rear < front)
      return;
    front++;
  }
  int query() {
    if (rear < front)
      return -1;
    return arr[front % SIZE];
  }
};
```

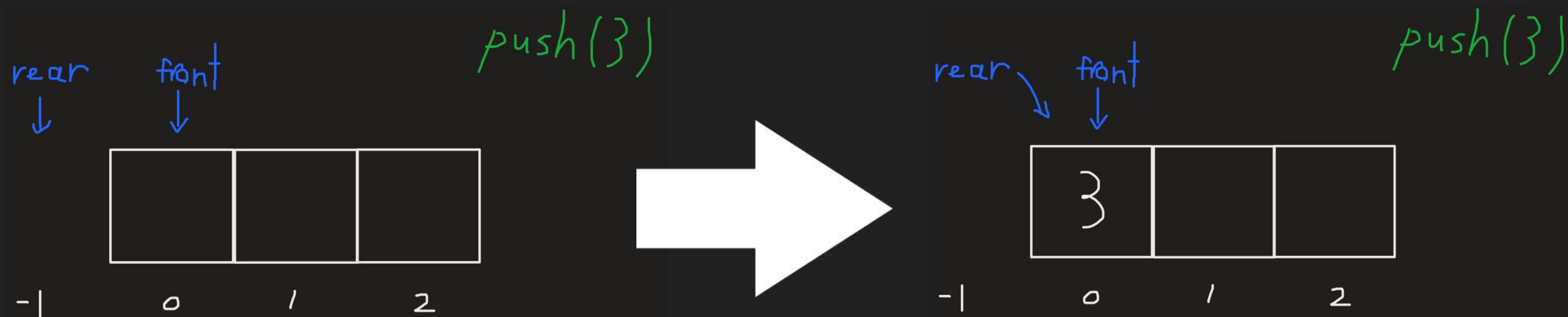
queue實作

為什麼要取模?

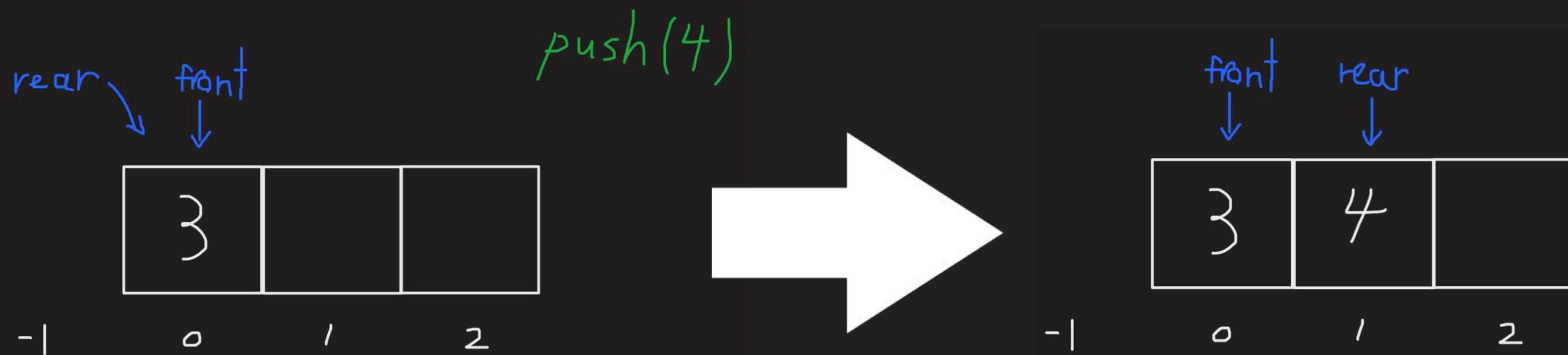
不取模會發生什麼事?

```
struct queue {
  static const int SIZE = 10000;
  int arr[SIZE];
  int front = 0;
  int rear = -1;
  void push(int val) {
    rear++;
    arr[rear % SIZE] = val;
  }
  void pop() {
    if (rear < front)
      return;
    front++;
  }
  int query() {
    if (rear < front)
      return -1;
    return arr[front % SIZE];
  }
};
```

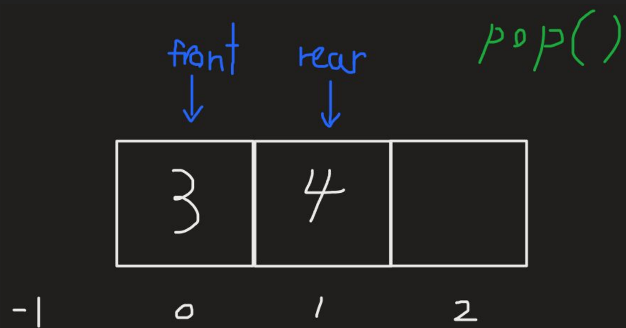
queue實作

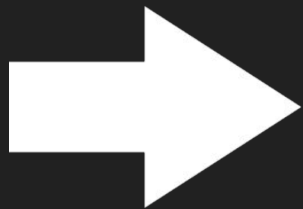


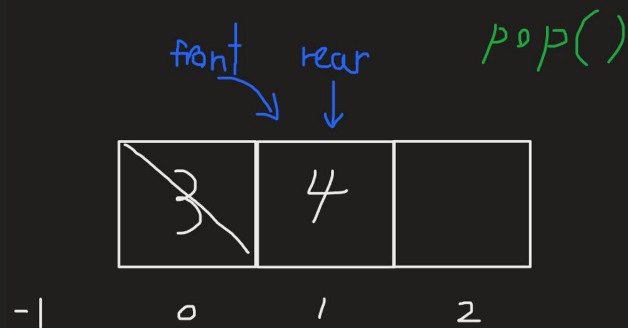
queue實作



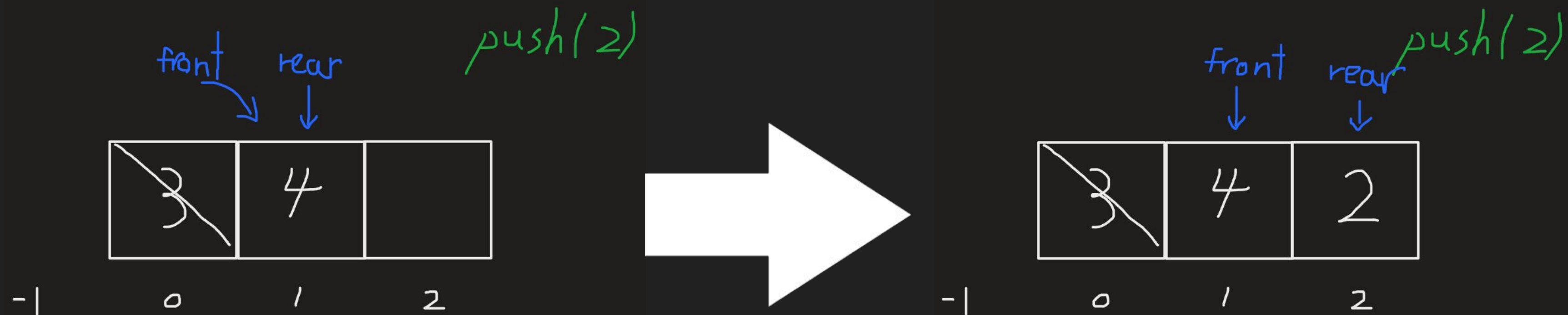
queue實作



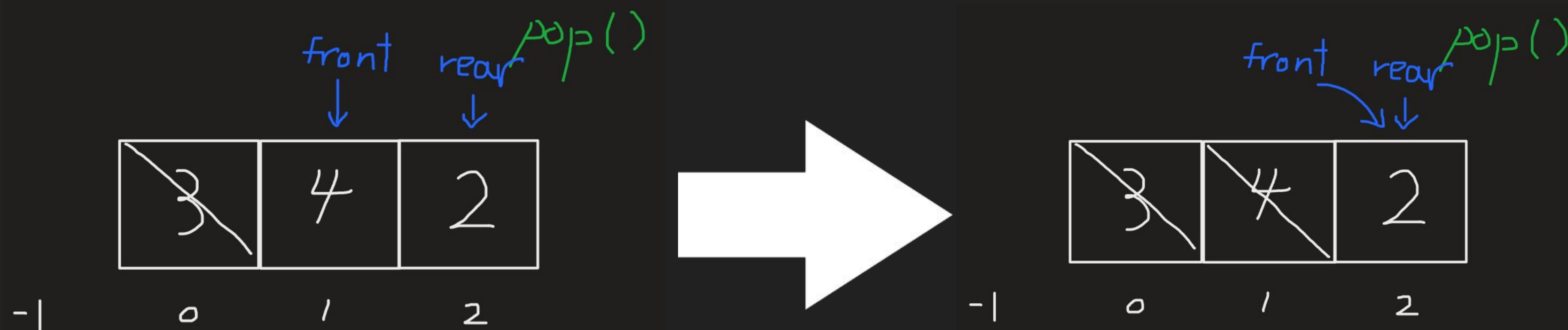




queue實作

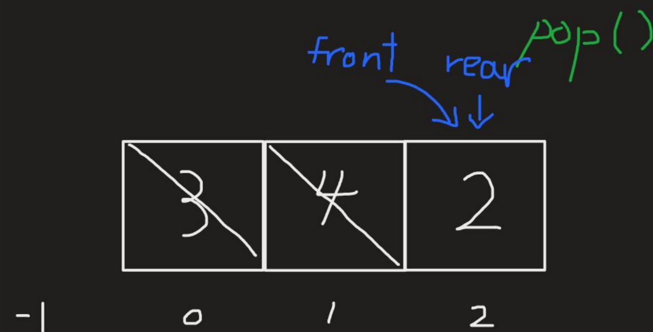


queue實作



queue實作

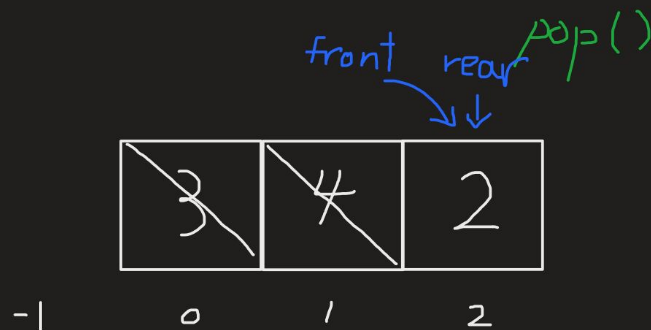
無法再繼續push元素進去了?



queue實作

無法再繼續push元素進去了？

這樣總共只能插入size個元素。

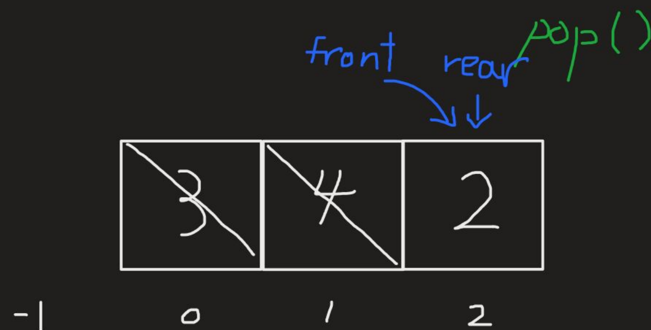


queue實作

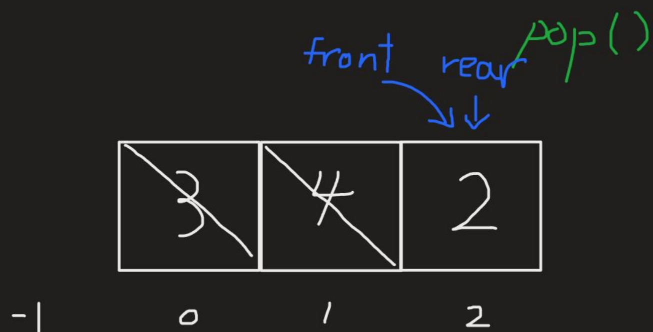
無法再繼續push元素進去了?

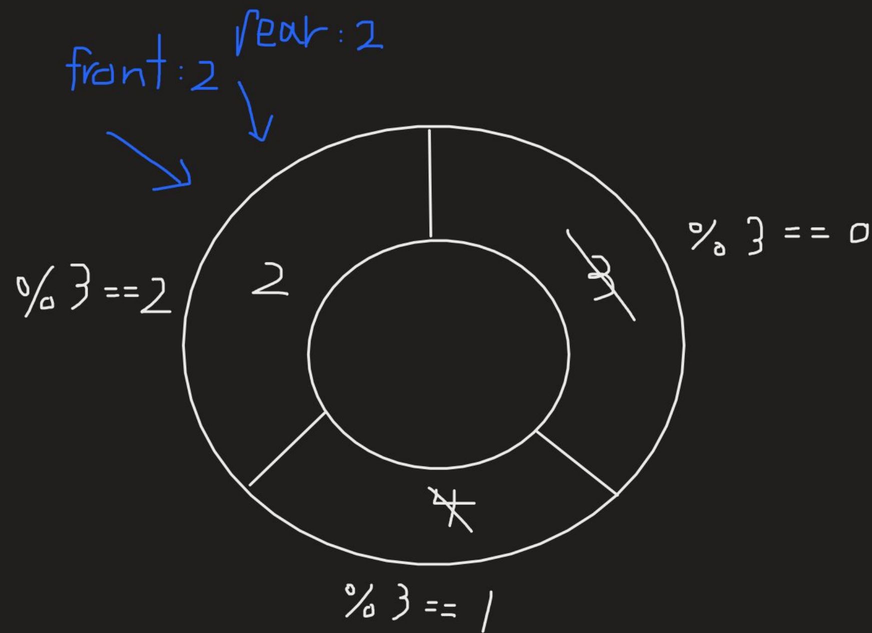
這樣總共只能插入size個元素。

取模?

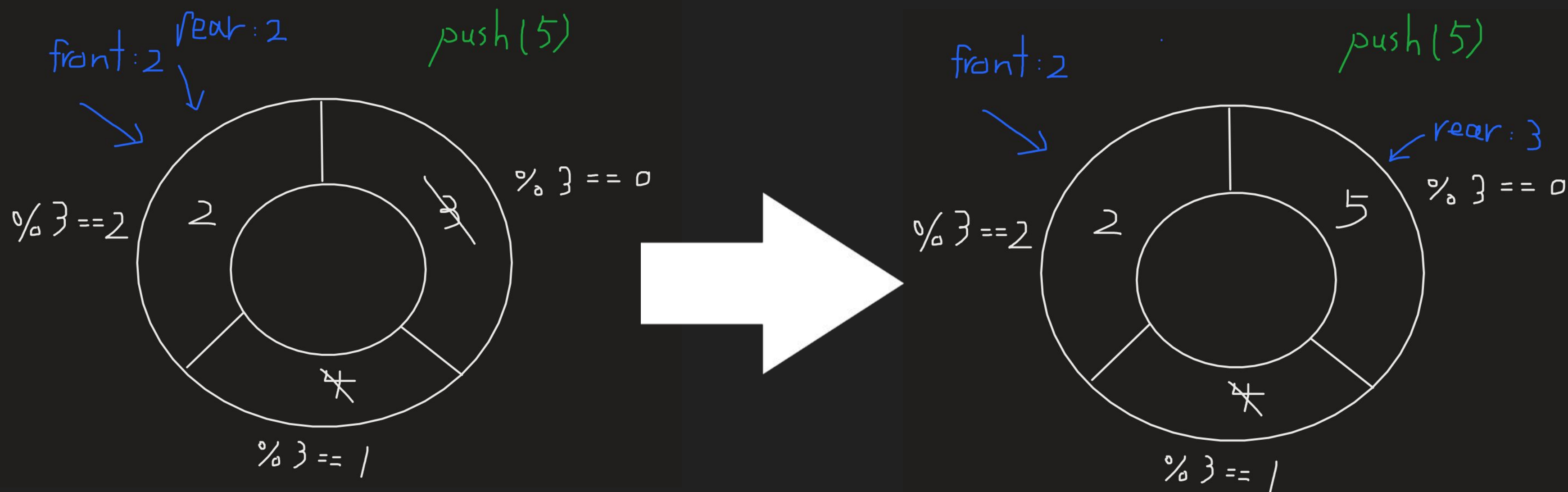


環狀queue





環狀queue



環狀queue

利用取模可以把queue變成環狀，讓可插入的元素總數不受限制，

但同時在queue裡面的元素還是不能超過size個。

linked list(鏈結串列)

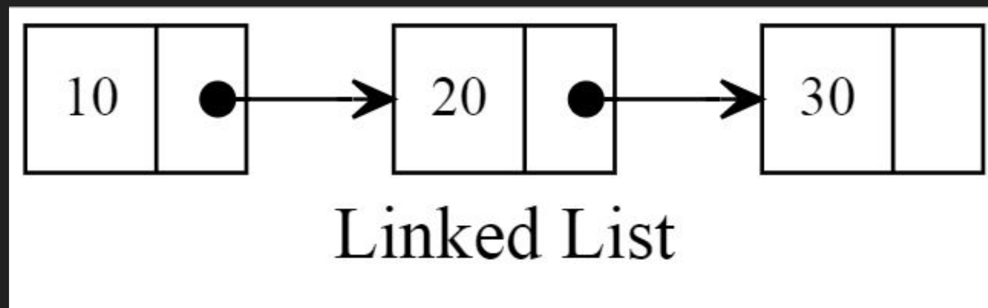
有時候我們會想要在陣列的中間插入一個元素,

但是如果我們直接把所有元素往後移動一格再插入,

這樣的複雜度會是O(n),

而linked list就是為了**快速插入/移除元素**而誕生的資料結構,

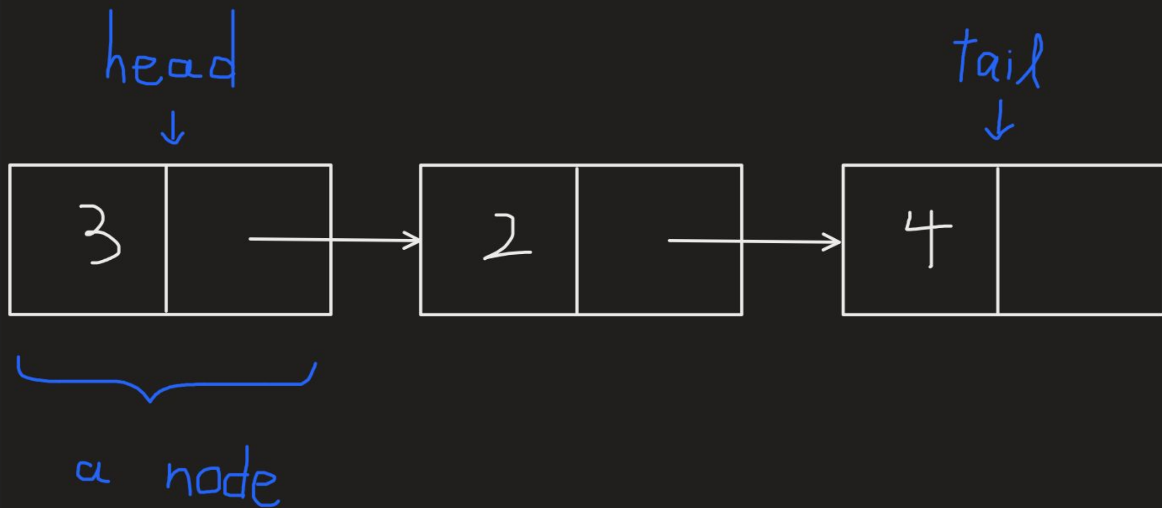
他的插入/移除都是O(1), 但缺點是不能隨機存取(random access)。



linked list(鏈結串列)

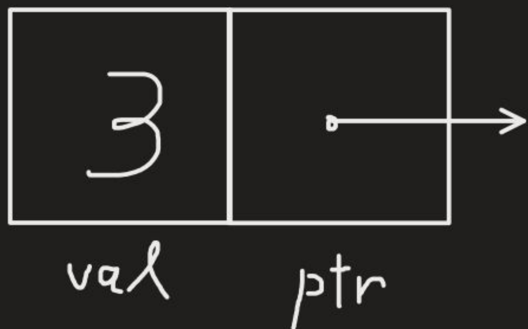
一個鏈結串列由很多節點(node)構成,

加上紀錄頭/尾節點的指標、當前節點的指標。



linked list的基本單位-node(節點)

一個節點有一格存元素的值，另一格用指標存下一個節點的位置。




```
struct node {
  int val;
  node* nxt = nullptr;
  node(int _val) : val(_val) {}
};
```

linked list基本結構

簡單來講就是用node彼此連接,

維護list的頭(head)跟尾(tail)及一個尋訪用的指針(cur)。

詳細實作建議在理解運作原理後自己刻刻看,

才會比較知道有什麼特殊的case要額外處理。

```
struct linkedList {
  node* head = nullptr;
  node* tail = nullptr;
  node* cur = nullptr;
  void insert(int val) {  //insert after cur
    if (!head) {
      head = tail = cur = new node(val);
    } else {
      node* tmp = cur -> nxt;
      cur -> nxt = new node(val);
      if (cur == tail)
        tail = cur -> nxt;
      cur -> nxt -> nxt = tmp;
    }
  }
  bool erase() {  //erase after cur
    if (!cur or !(cur -> nxt))
      return false;

    node* tmp = cur -> nxt -> nxt;
    if (cur -> nxt == tail)
      tail = cur;
    delete cur -> nxt;
    cur -> nxt = tmp;
    return true;
  }
  bool next() {
    if (!cur or cur -> nxt == nullptr)
      return false;

    cur = cur -> nxt;
    return true;
  }
  void reset() {
    cur = head;
  }
  int getCur() {
    if (!cur)
      return -1;
    else
      return cur -> val;
  }
```

linked list更多的功能

linked list也可以O(1)地在頭尾增加節點/

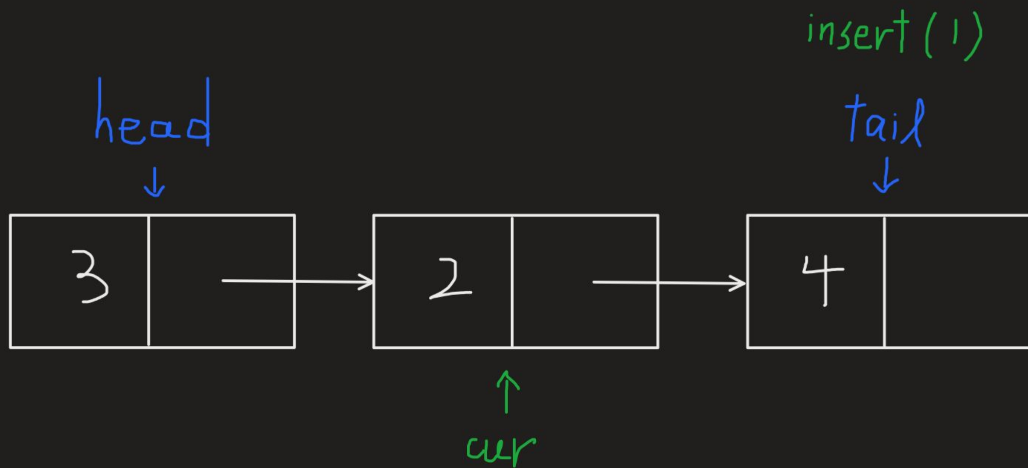
頭刪除節點。

所以前面所教的stack/queue

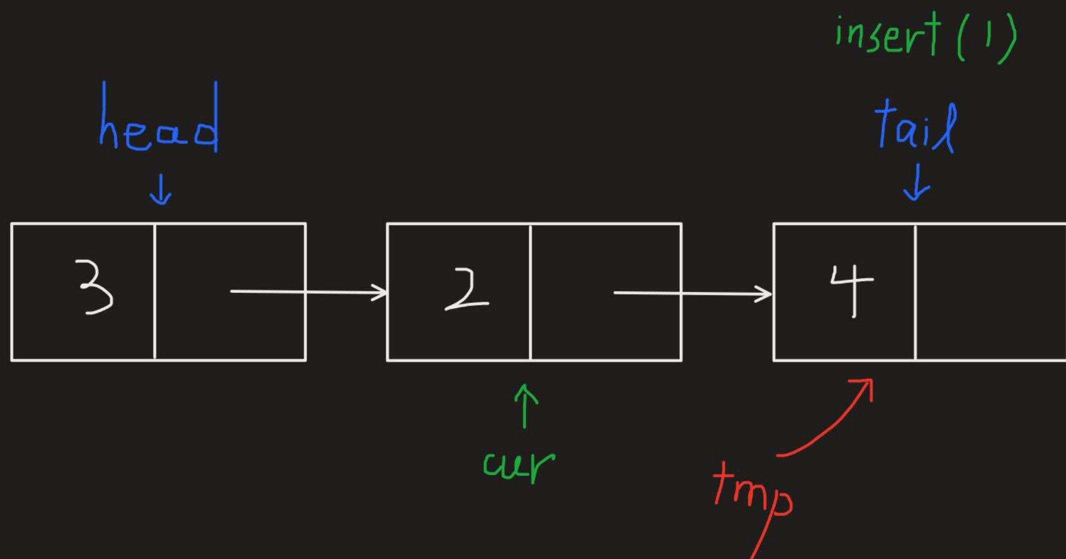
都可以用linked list實作。

```
  void push_front(int val) {
    if (!head) {
      head = tail = cur = new node(val);
    } else {
      node* tmp = head;
      head = new node(val);
      head -> nxt = tmp;
    }
  }
  bool pop_front() {
    if (!head)
      return false;

    if (head == tail)
      cur = tail = nullptr;
    else if (head == cur)
      cur = cur -> nxt;

    node* tmp = head -> nxt;
    delete head;
    head = tmp;
    return true;
  }
  void push_back(int val) {
    if (!tail) {
      head = tail = cur = new node(val);
    } else {
      tail -> nxt = new node(val);
      tail = tail -> nxt;
    }
  }
};
```

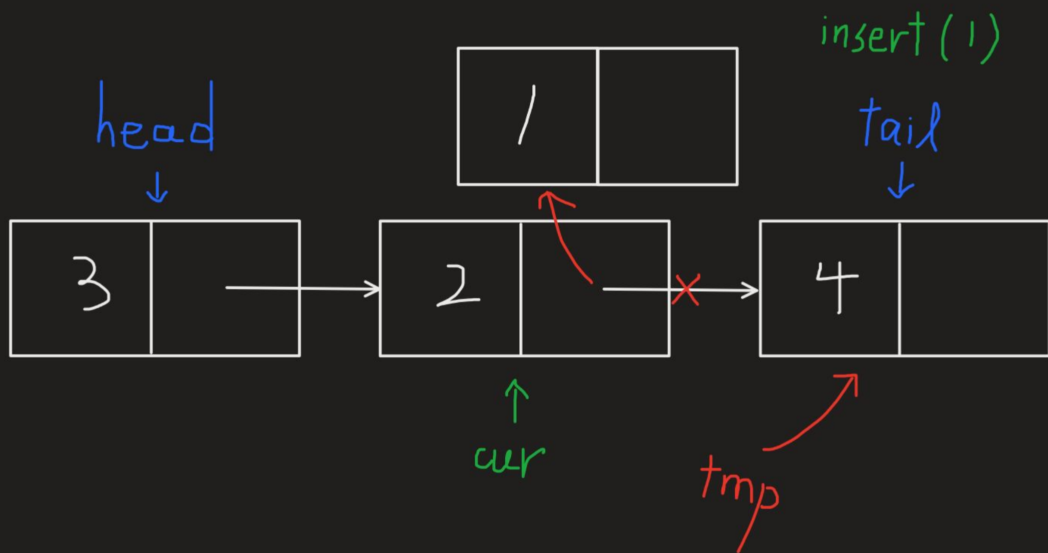
linked list支援的操作-插入(insert)



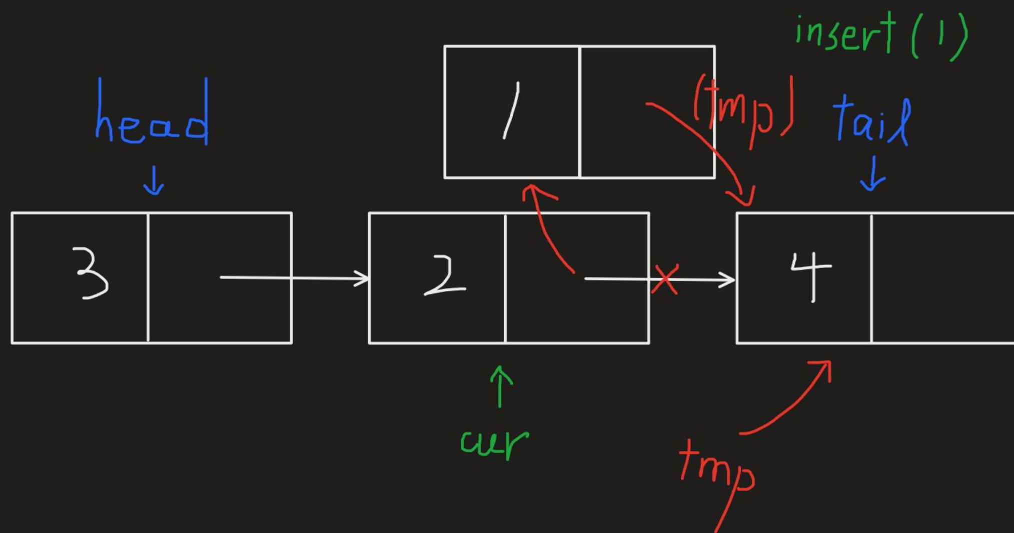
linked list支援的操作-插入(insert)



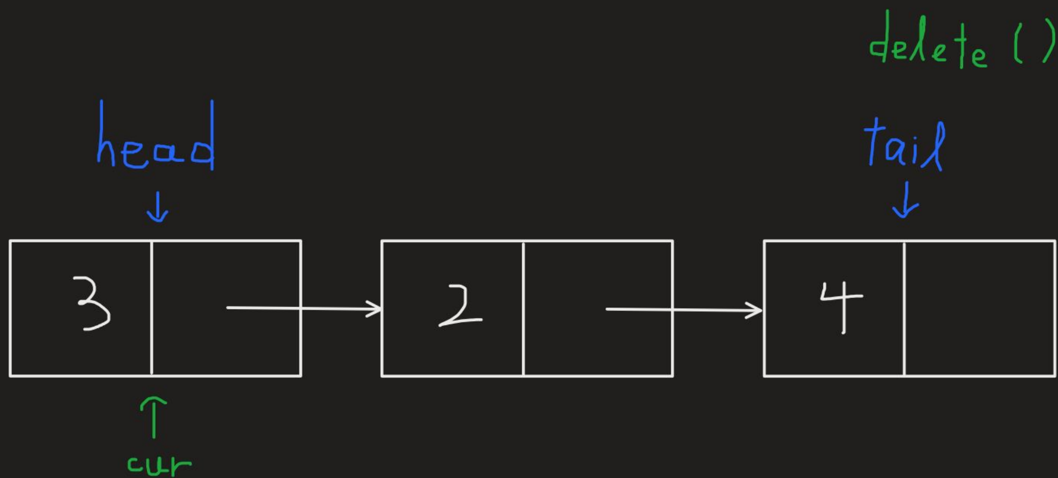
linked list支援的操作-插入(insert)



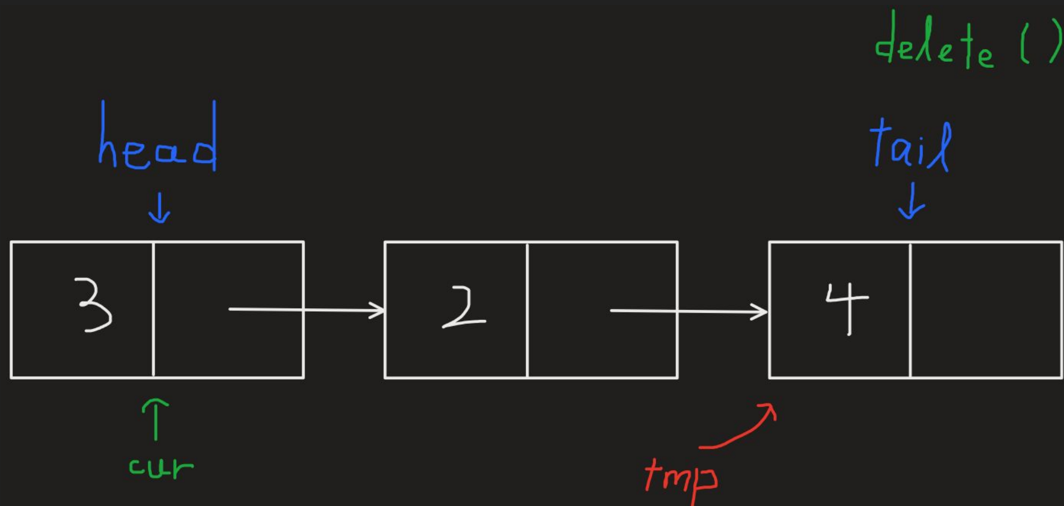
linked list支援的操作-插入(insert)



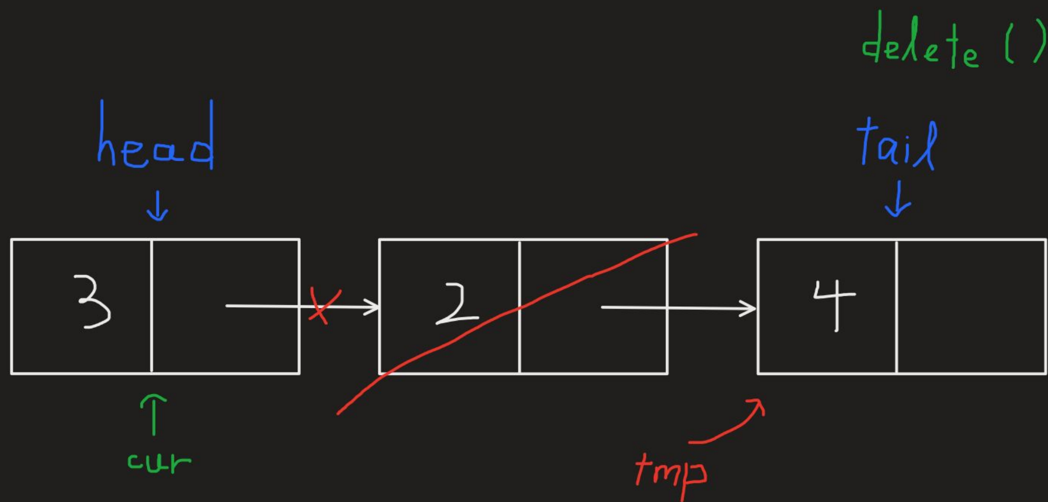
linked list支援的操作-刪除(delete)



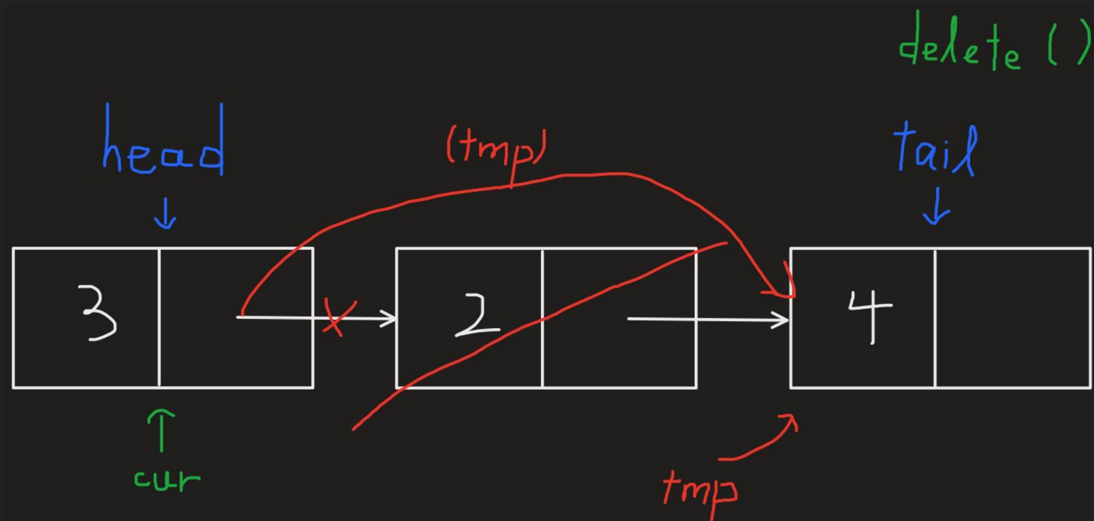
linked list支援的操作-刪除(delete)



linked list支援的操作-刪除(delete)



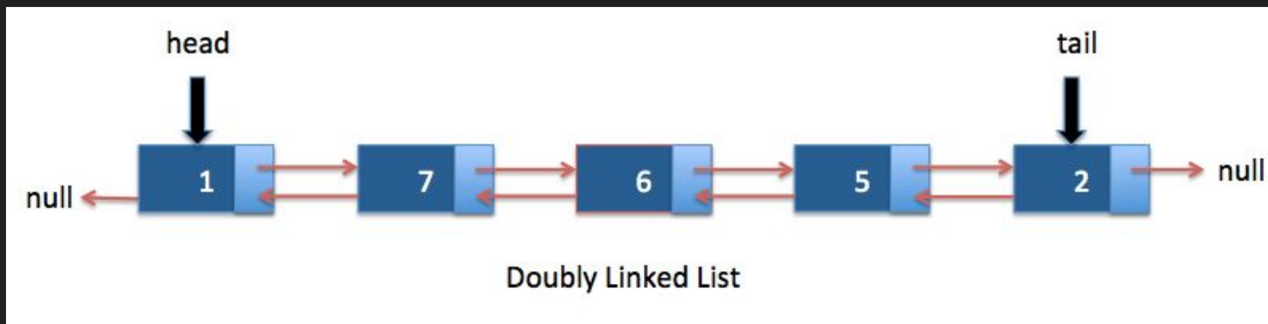
linked list支援的操作-刪除(delete)



各種linked list

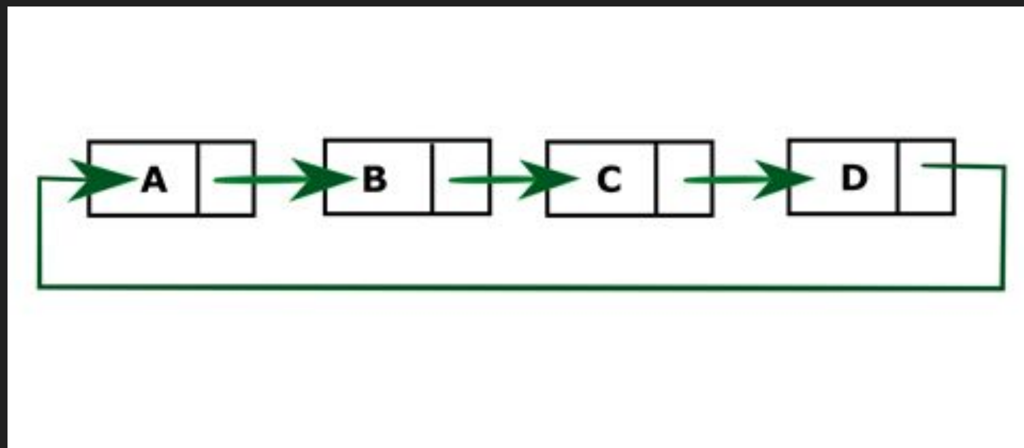
剛剛介紹的叫做單向linked list，cur指標只能往後跑，

要使cur能夠前後跑的話就要用雙向linked list(節點多存一個前一個節點的指標)。



各種linked list

如果把linked list的尾接到頭的話就是環狀linked list。



linked list的應用

1. 動態陣列
2. 實作queue/stack，不像陣列版會有最大大小的限制。

總結-各種資料結構的特色

stack: 先進後出(first in last out)

queue: 先進先出(first in first out)

linked list: 能快速的插入/刪除元素、不能隨機存取(random access)

ps. 這份內容的題單對應題目是pA, pD。