Data Structures

Lecture by qwe1rt1yuiop1

2025/02/22

Sprout

課程內容

- 什麼是資料結構？
- Stack
- Queue
- Deque
- 例題討論
- Linked List

Sprout

什麼是資料結構？

- 儲存資料的方式
- 快速從資料中找到特定資訊
  - 麻將：一眼看出有什麼牌型
  - 圖書館：快速找到書籍位置

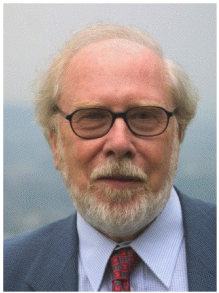
Sprout

什麼是資料結構？

- 儲存資料的方式
- 快速從資料中找到特定資訊
  - 麻將：一眼看出有什麼牌型
  - 圖書館：快速找到書籍位置
- 如何提取資訊？
- 如何維持資料？
- 好的資料處理方式，能讓程式節省時間與空間

Sprout

Why 資料結構？



Algorithms + Data Structures
= Programs

Niklaus Wirth,
the designer of Pascal

https://www.comp.hkbu.edu.hk/dlecture/wirth/nwprofile.php

Sprout

常見的資料結構

- Array
- Stack, Queue, Deque - today!
- Linked List - today!
- Set, Map
- Heap - week 3
- Binary Indexed Tree - hand 10
- Disjoint Set - hand 12
- .....

Sprout

Stack

Sprout

Stack 堆疊

https://pixabay.com/vectors/books-literature-pile-study-2022464/

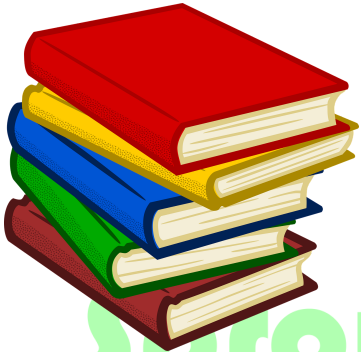
- 要怎麼拿到綠色的那本書？

Stack 堆疊

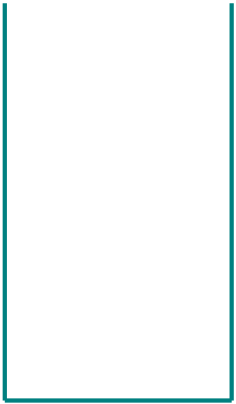
https://pixabay.com/vectors/books-literature-pile-study-2022464/

- 要怎麼拿到綠色的那本書？

依序把紅、黃、藍的書拿起來
拿到綠色的書
再依序將藍、黃、紅的書放回去

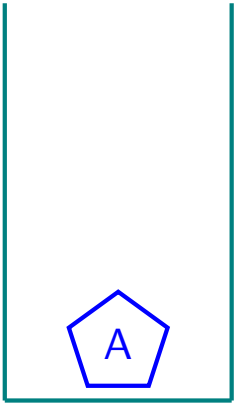


Stack 堆疊



Empty

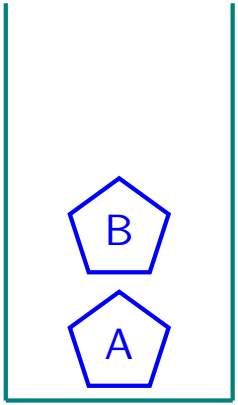
Stack 堆疊



Empty

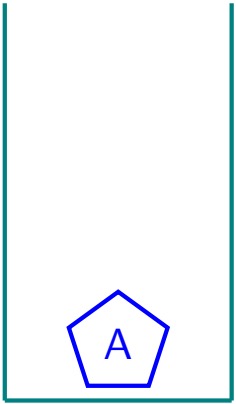
加入資料 A 到最頂端

Stack 堆疊



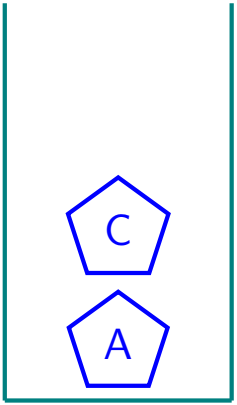
Empty
加入資料 A 到最頂端
加入資料 B 到最頂端

Stack 堆疊



Empty
加入資料 A 到最頂端
加入資料 B 到最頂端
刪除最頂端資料

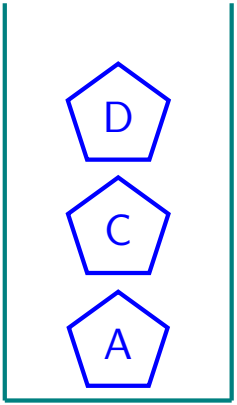
Stack 堆疊



Empty
加入資料 A 到最頂端
加入資料 B 到最頂端
刪除最頂端資料
加入資料 C 到最頂端

Sprout

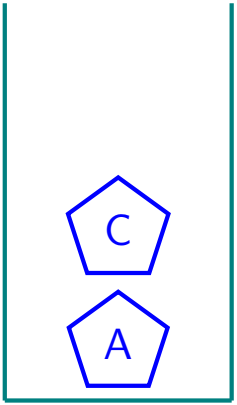
Stack 堆疊



Empty
加入資料 A 到最頂端
加入資料 B 到最頂端
刪除最頂端資料
加入資料 C 到最頂端
加入資料 D 到最頂端

Sprout

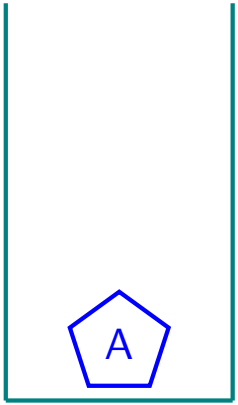
Stack 堆疊



Empty
加入資料 A 到最頂端
加入資料 B 到最頂端
刪除最頂端資料
加入資料 C 到最頂端
加入資料 D 到最頂端
刪除最頂端資料

Sprout

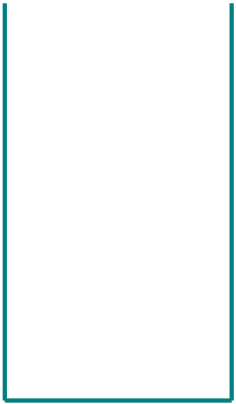
Stack 堆疊



Empty
加入資料 A 到最頂端
加入資料 B 到最頂端
刪除最頂端資料
加入資料 C 到最頂端
加入資料 D 到最頂端
刪除最頂端資料
刪除最頂端資料

Sprout

Stack 堆疊



Empty
加入資料 A 到最頂端
加入資料 B 到最頂端
刪除最頂端資料
加入資料 C 到最頂端
加入資料 D 到最頂端
刪除最頂端資料
刪除最頂端資料
刪除最頂端資料

Sprout

Stack 的特性

- 操作：
  - 新增資料到最頂端
  - 刪除最頂端的資料
  - 存取最頂端的資料
- 先進後出 (First In Last Out, FILO)
- 後進先出 (Last In First Out, LIFO)

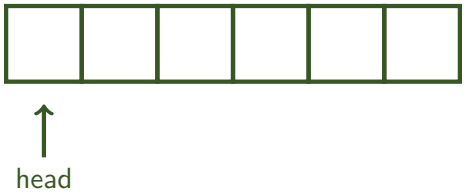
Sprout

Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小

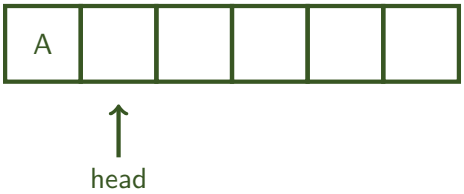
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



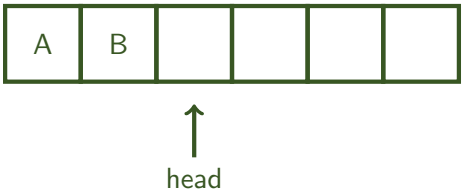
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



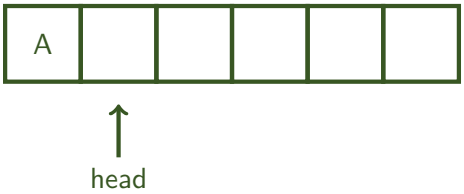
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



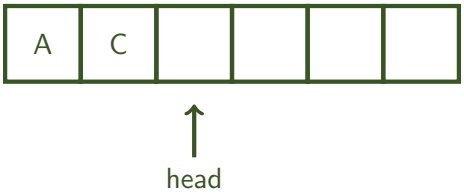
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



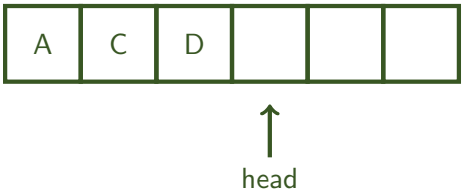
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



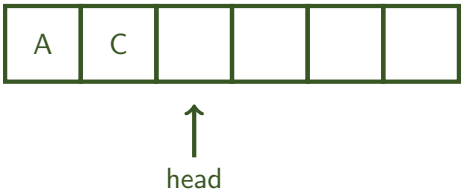
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



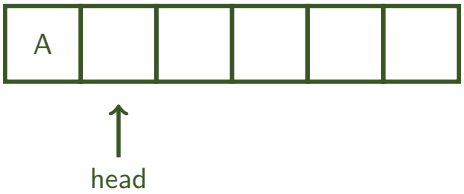
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



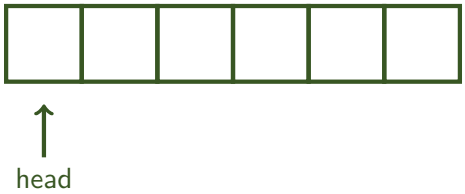
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



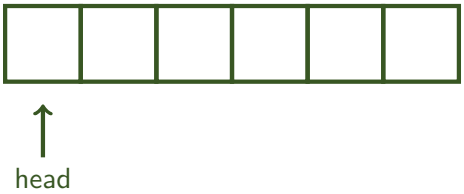
Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的



Stack 的實作

- `push()` 加入一筆資料到最頂端
- `pop()` 刪除最頂端的資料
- `top()` 回傳最頂端的資料
- `size()` 回傳 stack 的大小
- 用陣列代表 stack
- 用變數 head 記錄頂端位置
  - head $= 0$ 代表 stack 是空的
- 陣列要開多大？
  - 最多可能同時幾筆資料就開多大
  - 不會有浪費的記憶體



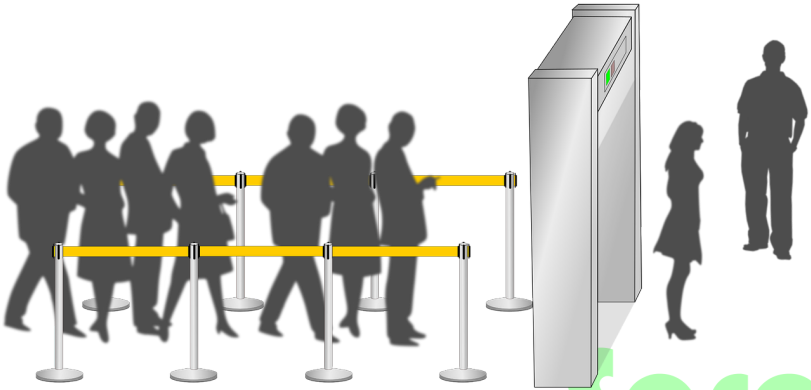

```
struct Stack {
    int arr[MAXN], head;
    Stack() : head(0) {}
    void push(int val) {
        arr[head++] = val;
    }
    void pop() {
        head--;
    }
    int top() {
        return arr[head - 1];
    }
    int size() {
        return head;
    }
};
```

Sprout

Queue

Sprout

Queue 佇列



https://pixabay.com/vectors/retractable-belt-stanchion-crowd-4326819/

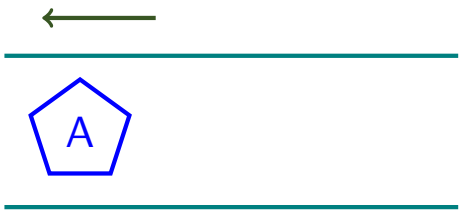
Queue 佇列

Empty

Queue 佇列

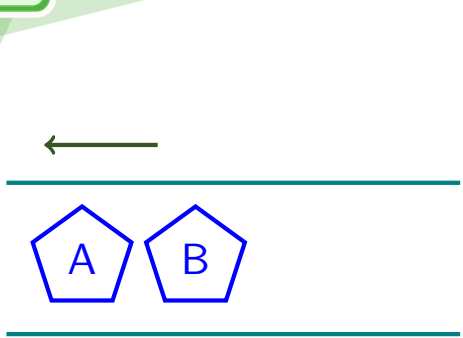
Empty

加入資料 A 到最後端

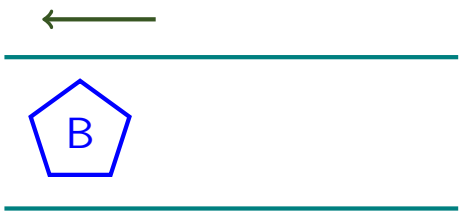


Queue 佇列

Empty
加入資料 A 到最後端
加入資料 B 到最後端

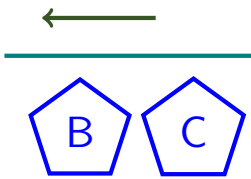


Queue 佇列



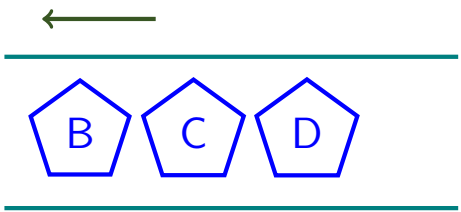
Empty
加入資料 A 到最後端
加入資料 B 到最後端
刪除最前端資料

Queue 佇列



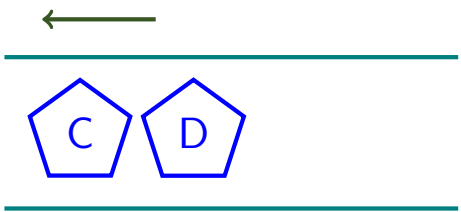
Empty
加入資料 A 到最後端
加入資料 B 到最後端
刪除最前端資料
加入資料 C 到最後端

Queue 佇列



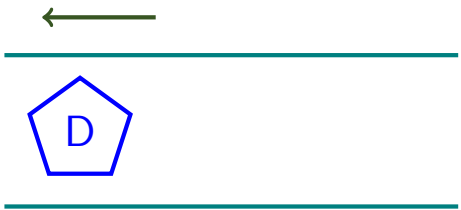
Empty
加入資料 A 到最後端
加入資料 B 到最後端
刪除最前端資料
加入資料 C 到最後端
加入資料 D 到最後端

Queue 佇列



Empty
加入資料 A 到最後端
加入資料 B 到最後端
刪除最前端資料
加入資料 C 到最後端
加入資料 D 到最後端
刪除最前端資料

Queue 佇列



Empty
加入資料 A 到最後端
加入資料 B 到最後端
刪除最前端資料
加入資料 C 到最後端
加入資料 D 到最後端
刪除最前端資料
刪除最前端資料

Queue 佇列

Empty
加入資料 A 到最後端
加入資料 B 到最後端
刪除最前端資料
加入資料 C 到最後端
加入資料 D 到最後端
刪除最前端資料
刪除最前端資料
刪除最前端資料

Queue 的特性

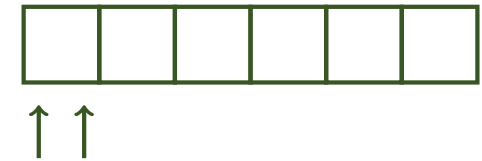
- 操作：
  - 新增資料到最後端
  - 刪除最前端的資料
  - 存取最前端的資料
- 先進先出 (First In First Out, FIFO)
- 後進後出 (Last In Last Out, LILO)

Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小

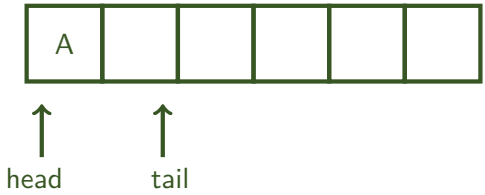
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



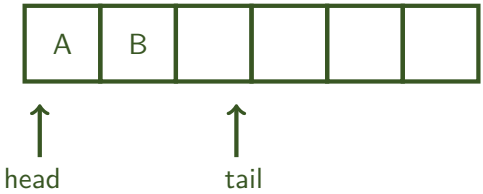
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



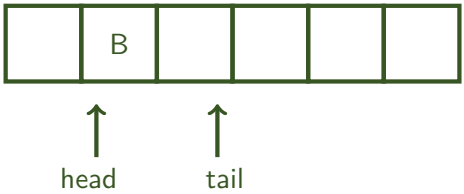
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



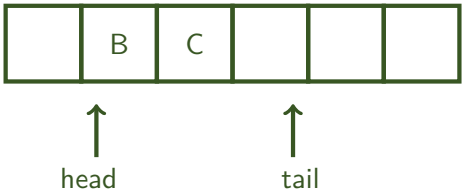
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



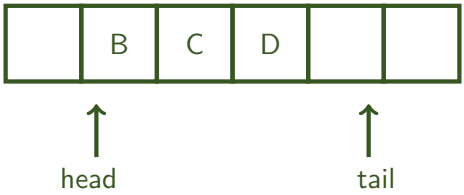
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



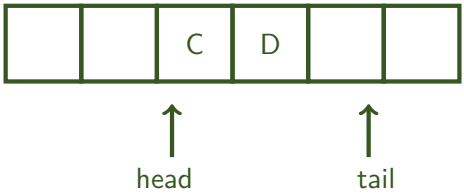
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



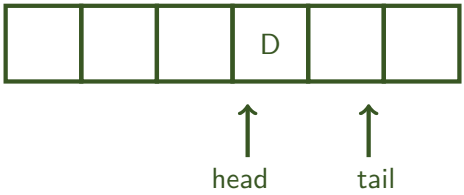
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



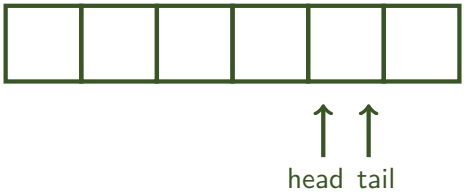
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



Queue 的實作

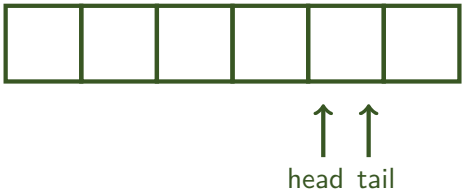
- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的



Sprout

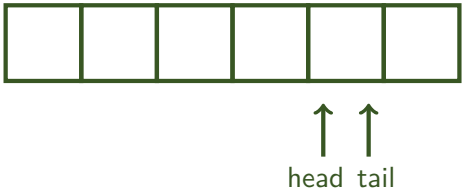
Queue 的實作

- `push()` 加入一筆資料到最後端
- `pop()` 刪除最前端的資料
- `front()` 回傳最前端的資料
- `size()` 回傳 queue 的大小
- 用陣列代表 queue
- 用變數 head、tail 記錄前後端
  - head = tail 代表 queue 是空的
- 陣列要開多大？
  - 丟掉的東西還是佔空間...



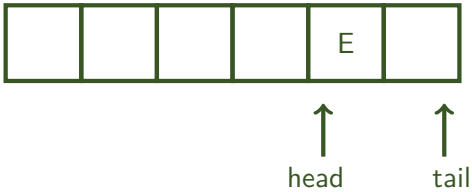
Circular Queue

- 碰到陣列尾巴再繞回來



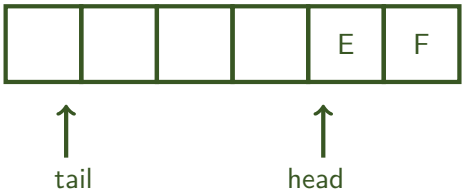
Circular Queue

- 碰到陣列尾巴再繞回來



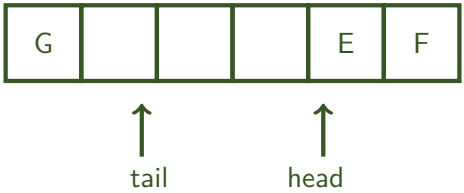
Circular Queue

- 碰到陣列尾巴再繞回來



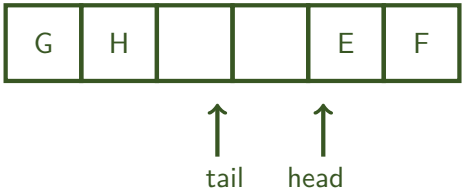
Circular Queue

- 碰到陣列尾巴再繞回來



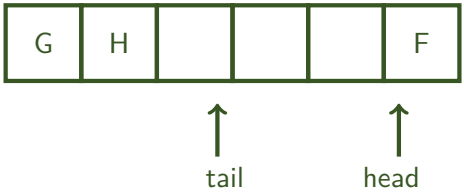
Circular Queue

- 碰到陣列尾巴再繞回來



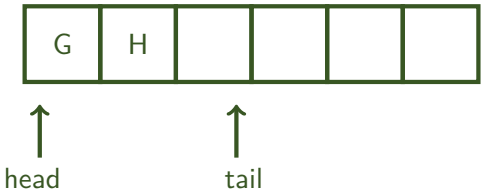
Circular Queue

- 碰到陣列尾巴再繞回來



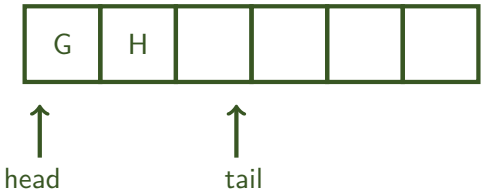
Circular Queue

- 碰到陣列尾巴再繞回來



Circular Queue

- 碰到陣列尾巴再繞回來
- 當 queue 大小上限不高，
  可以避免記憶體空間浪費的問題




```
struct Queue {
    int arr[MAXN], head, tail;
    Queue() : head(0), tail(0) {}
    int front() {
        return arr[head];
    }
    void pop() {
        head = (head + 1) % MAXN;
    }
    void push(int val) {
        arr[tail] = val;
        tail = (tail + 1) % MAXN;
    }
    int size() {
        return (tail - head + MAXN) % MAXN;
    }
};
```

Deque

Sprout

Deque 雙端佇列

- Double-ended Queue
- 有些人喜歡念成「de-queue」
- A deque ("double-ended queue") is a linear list for which all insertions and deletions (and usually all accesses) are made at the ends of the list. A deque is therefore more general than a stack or a queue; it has some properties in common with a deck of cards, and it is pronounced the same way.

  The Art of Computer Programming

Deque 的實作

- `push_front()`, `push_back()` 加入前端 / 後端
- `pop_front()`, `pop_back()` 刪除前端 / 後端
- `front()`, `back()` 詢問前端 / 後端
- `size()` 詢問大小

Sprout

Deque 的實作

- `push_front()`, `push_back()` 加入前端 / 後端
- `pop_front()`, `pop_back()` 刪除前端 / 後端
- `front()`, `back()` 詢問前端 / 後端
- `size()` 詢問大小
- 用陣列代表 deque
- 用變數 head、tail 記錄前後端
  - head = tail 代表 deque 是空的
  - 依照操作調整 head / tail

例題討論

Sprout

括弧匹配

Problem 括弧匹配

給定一個僅包含（或）的字串，長度為 N，問其是否為合法括弧字串，是的話也請找到所有配對，依序輸出每個括弧的配對對象位置。

- $1 \leq N \leq 10^6$

舉例來說，()(()()) 是一個合法括弧字串，而)((()() 不是合法括弧字串。

括弧匹配

- 什麼樣的字串是合法括弧字串？
  - 長度偶數，左右括弧數量相等
  - 每個 **右括弧** 都能往 **左邊** 找到對應的 **左括弧**
  - 每個 **左括弧** 都能往 **右邊** 找到對應的 **右括弧**

括弧匹配

- 什麼樣的字串是合法括弧字串？
  - 長度偶數，左右括弧數量相等
  - 每個 **右括弧** 都能往 **左邊** 找到對應的 **左括弧**
  - 每個 **左括弧** 都能往 **右邊** 找到對應的 **右括弧**
- 對應是什麼意思？
  - 要在同一層才能對應 $\rightarrow$ 關注每個括弧在哪一層

括弧匹配

- 什麼樣的字串是合法括弧字串？
  - 長度偶數，左右括弧數量相等
  - 每個 **右括弧** 都能往 **左邊** 找到對應的 **左括弧**
  - 每個 **左括弧** 都能往 **右邊** 找到對應的 **右括弧**
- 對應是什麼意思？
  - 要在同一層才能對應 $\rightarrow$ 關注每個括弧在哪一層
  - 層數初始為 0，從左往右掃，左括弧會讓層數 $+1$，右括弧則是 -1
  - 已經掃過多少未配對的左括弧

括弧匹配

- 什麼樣的字串是合法括弧字串？
  - 長度偶數，左右括弧數量相等
  - 每個 **右括弧** 都能往 **左邊** 找到對應的 **左括弧**
  - 每個 **左括弧** 都能往 **右邊** 找到對應的 **右括弧**
- 對應是什麼意思？
  - 要在同一層才能對應 → 關注每個括弧在哪一層
  - 層數初始為 0，從左往右掃，左括弧會讓層數 +1，右括弧則是 -1
  - 已經掃過多少未配對的左括弧
- 合法括弧字串要滿足的條件
  - 過程中不能有負數層
  - 最後層數要回到 0

括弧匹配

- 要怎麼知道誰跟誰配對？

括弧匹配

- 要怎麼知道誰跟誰配對？
- 層數遇到右括弧會 -1，產生了一組配對。是跟哪個左括號配對？
- 用資料結構把未配對的左括弧 index 存起來

Sprout

括弧匹配

- 要怎麼知道誰跟誰配對？
- 層數遇到右括弧會 -1，產生了一組配對。是跟哪個左括號配對？
- 用資料結構把未配對的左括弧 index 存起來
  - Stack!
  - 層數就是 stack 的 size

Sprout

括弧匹配

- 要怎麼知道誰跟誰配對？
- 層數遇到右括弧會 -1，產生了一組配對。是跟哪個左括號配對？
- 用資料結構把未配對的左括弧 index 存起來
  - Stack!
  - 層數就是 stack 的 size
- 左括弧：直接 push，層數 $+1$
- 右括弧：stack 是空的就失敗，否則讓它跟 top 配對後 pop，層數 -1
- 最後檢查 stack 是不是空的

Nearest Smaller Value

Problem Nearest Smaller Value[1]

給定長度為 N 的序列 $a_1, a_2, ..., a_N$，對於每個數字，請找到它左邊第一個比它小的數字，輸出該數字的位置。若找不到請輸出 0。

- $1 \leq N \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

[1] CSES 1645

Nearest Smaller Value

- 從左往右掃，設法快速找到目前數字的答案（從已掃過的找）
- 有些候選人以後都不可能是別人的答案！

Observation 單調性

對於位置 $i < j$，若有 $a_i \geq a_j$，對所有的 $k \geq j$ 來說，i 都不可能是 k 的答案。

Sprout

Nearest Smaller Value

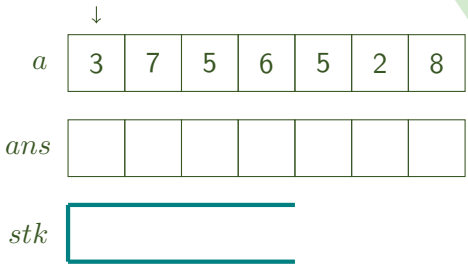
- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index

Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構

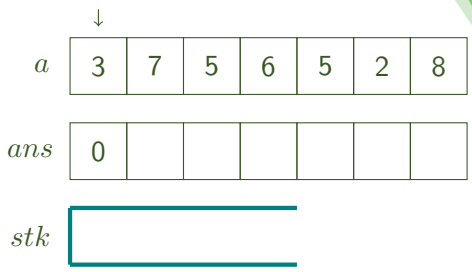
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



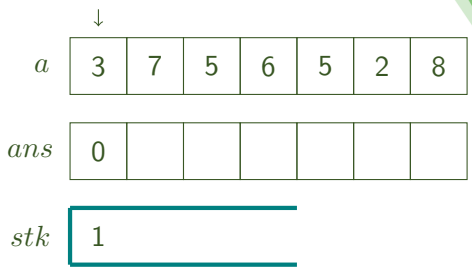
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



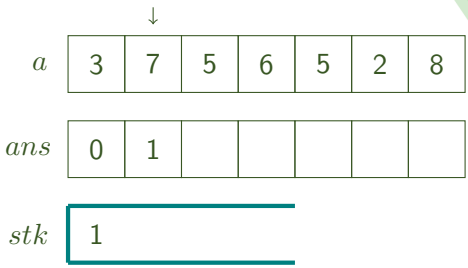
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



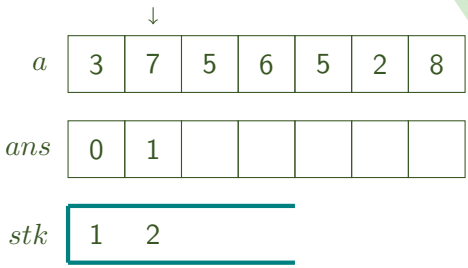
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



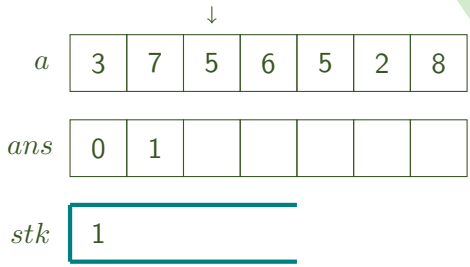
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



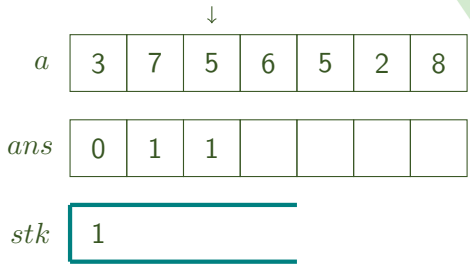
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



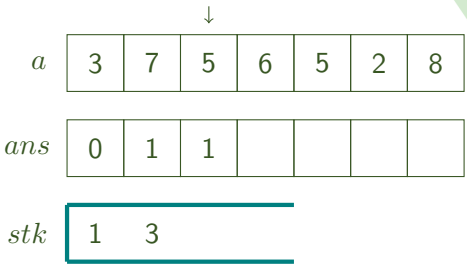
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



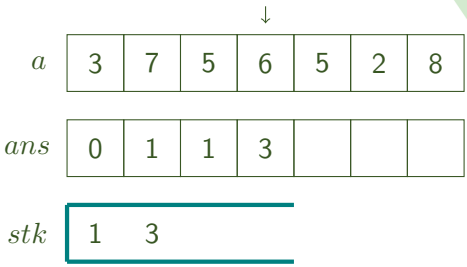
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



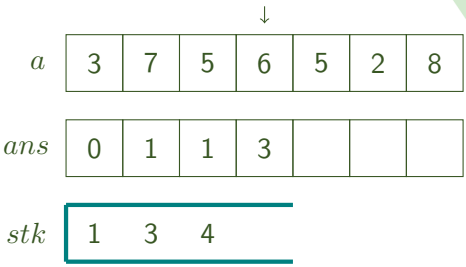
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



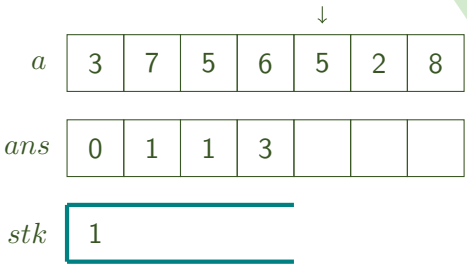
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



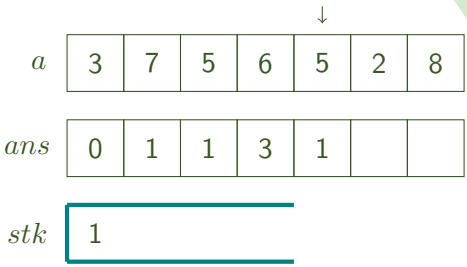
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



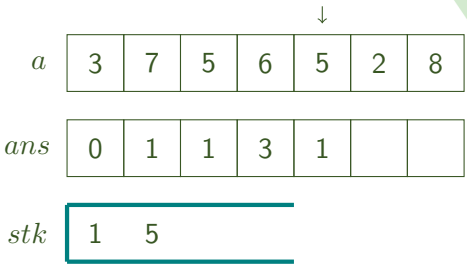
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



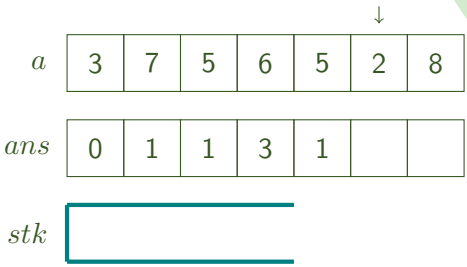
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



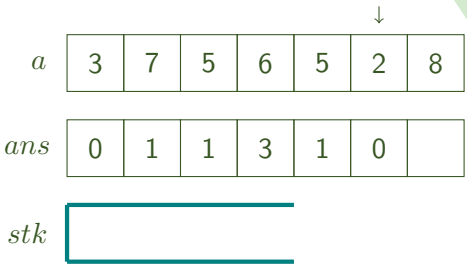
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



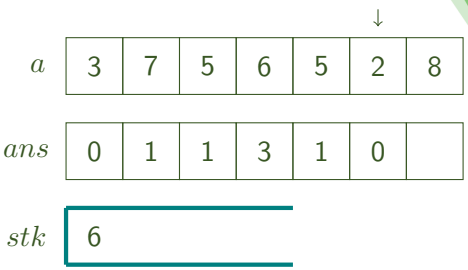
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



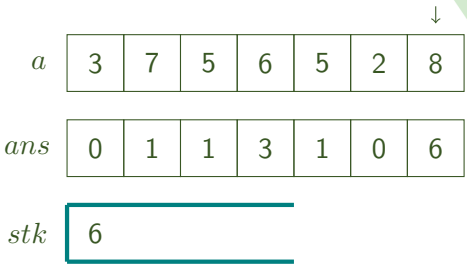
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



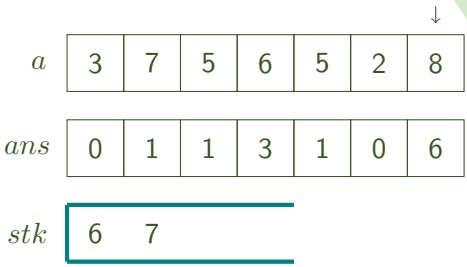
Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!



Nearest Smaller Value

- 考慮哪些人還有可能是以後的答案
  - 這些值會是嚴格遞增的
  - 用資料結構存它們的 index
- 掃到某個新的數字時
  1. 它會讓某些可能是答案的，往後不再可能是答案。把這些丟掉
  2. 找到這個數字的答案
  3. 它永遠都有可能會是以後的答案，加入資料結構
- Stack!




```
for (int i = 1; i <= n; ++i) {
    while (stk.size() > 0 &&
            a[stk.top()] >= a[i])
        stk.pop();
    if (stk.size() == 0)
        ans[i] = 0;
    else ans[i] = stk.top();
    stk.push(i);
}
```

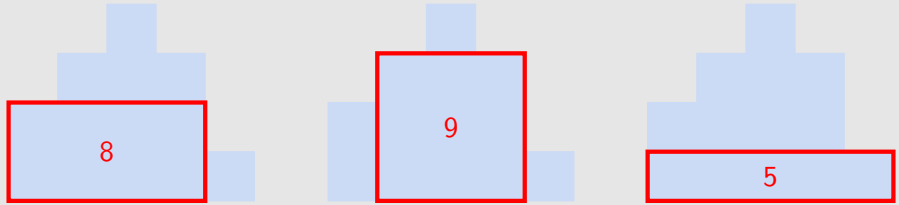
Sprout

長條圖最大矩形

Problem 長條圖最大矩形[2]

給定長度為 N 的序列 $a_1, a_2, ..., a_N$，代表等寬長條圖每個位置的高度，請找到能畫在長條圖範圍內的矩形最大面積。

- $1 \leq N \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$



[2]Sprout OJ 425 (NEOJ 513)

長條圖最大矩形

- 直覺的作法
  - 枚舉 $O(N^2)$ 個區間
  - 高度至多只能到最矮的那個，$O(N)$ 掃一遍區間找最小值

Sprout

長條圖最大矩形

- 直覺的作法
  - 枚舉 $O(N^2)$ 個區間
  - 高度至多只能到最矮的那個，$O(N)$ 掃一遍區間找最小值
- 「一段區間的高度至多只能到最矮的那個」

長條圖最大矩形

- 直覺的作法
  - 枚舉 $O(N^2)$ 個區間
  - 高度至多只能到最矮的那個，$O(N)$ 掃一遍區間找最小值
- 「一段區間的高度至多只能到最矮的那個」
- 重點不是區間，是「最矮的那個」
  - 從枚舉區間，變成枚舉每個 bar 的高度

長條圖最大矩形

- 直覺的作法
  - 枚舉 $O(N^2)$ 個區間
  - 高度至多只能到最矮的那個，$O(N)$ 掃一遍區間找最小值
- 「一段區間的高度至多只能到最矮的那個」
- 重點不是區間，是「最矮的那個」
  - 從枚舉區間，變成枚舉每個 bar 的高度
- 如果我是最低的，那往左往右至多可以延伸多少？
  - 分別找到左右兩邊第一個比我小的！

長條圖最大矩形

步驟整理

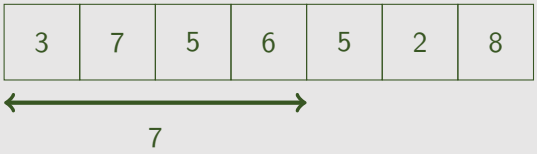
1. 要找最大矩形，可以枚舉每個值作為最小值時的情況
2. 將向左、向右拆開成兩個問題
3. 用單調 stack 解「找到左 / 右邊第一個比我小的值」
4. 合併左右算出答案，對所有枚舉的情況取最大值即所求

延伸問題

Problem Sliding Window[3]

給定長度為 N 的序列 $a_1, a_2, ..., a_N$ 和一個整數 K，問每個長度 K 連續區間的區間最大值。

- $1 \leq K \leq N \leq 10^6$
- $1 \leq a_i \leq 10^9$



[3] Zerojudge a146

延伸問題

Problem Sliding Window[3]

給定長度為 N 的序列 $a_1, a_2, ..., a_N$ 和一個整數 K，問每個長度 K 連續區間的區間最大值。

- $1 \leq K \leq N \leq 10^6$
- $1 \leq a_i \leq 10^9$

[3] Zerojudge a146

延伸問題

Problem Sliding Window[3]

給定長度為 N 的序列 $a_1, a_2, ..., a_N$ 和一個整數 K，問每個長度 K 連續區間的區間最大值。

- $1 \leq K \leq N \leq 10^6$
- $1 \leq a_i \leq 10^9$

[3] Zerojudge a146

延伸問題

Problem Sliding Window[3]

給定長度為 N 的序列 $a_1, a_2, ..., a_N$ 和一個整數 K，問每個長度 K 連續區間的區間最大值。

- $1 \leq K \leq N \leq 10^6$
- $1 \leq a_i \leq 10^9$

[3] Zerojudge a146

Linked List

Sprout

Linked List 鏈結串列

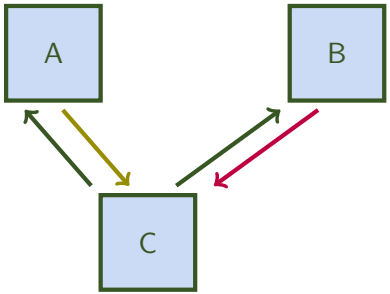
- 對每筆資料紀錄前、後分別是哪筆資料（或沒有）
- 可以 $O(1)$ 對某筆資料進行加入、刪除等操作
- 無法支援 random-access
  - 不能 $O(1)$ 存取 linked list 裡的第 i 筆資料

Linked List 鏈結串列

- 將資料 C 插入在資料 A、B 之間（反操作就是刪除）

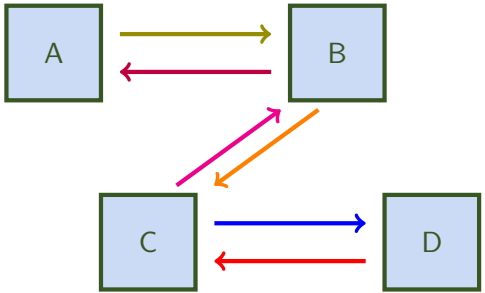
Linked List 鏈結串列

- 將資料 C 插入在資料 A、B 之間（反操作就是刪除）



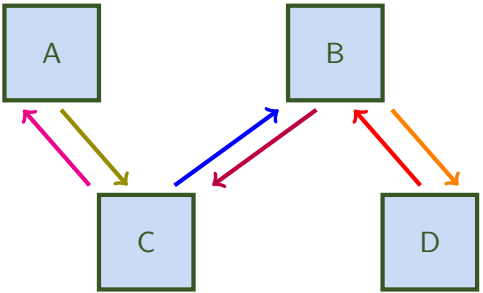
Linked List 鏈結串列

- 將資料 B、C 換位置



Linked List 鏈結串列

- 將資料 B、C 換位置



謝謝大家！

Sprout