



# Data Structures

Lecture by Jyun-An Chen (Colten)

Credit by Casper Wang

2024.02.24 Algorithm-Tainan

**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？

Sprout



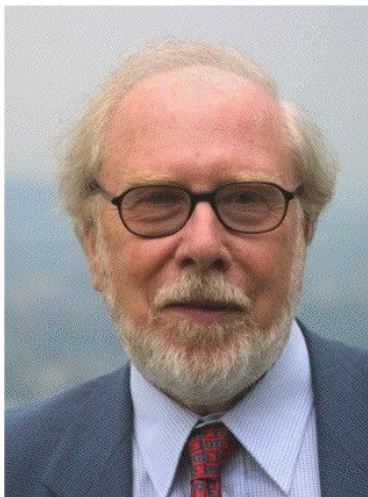
## 資料結構

- 是電腦中儲存、組織資料的方式
- 好的資料處理方式，能讓程式節省時間與空間
- 例如：「陣列」就是一種資料結構

Sprout



## Why 資料結構？



Algorithms + Data Structures  
= Programs

Niklaus Wirth,  
the designer of Pascal

Sprout



## 常見的資料結構

- Stack, Queue, Deque
- Linked-list
- Heap
- Set, Map
- Disjoint Set
- Bit, Segment Tree, Sparse Table
- .....

Sprout



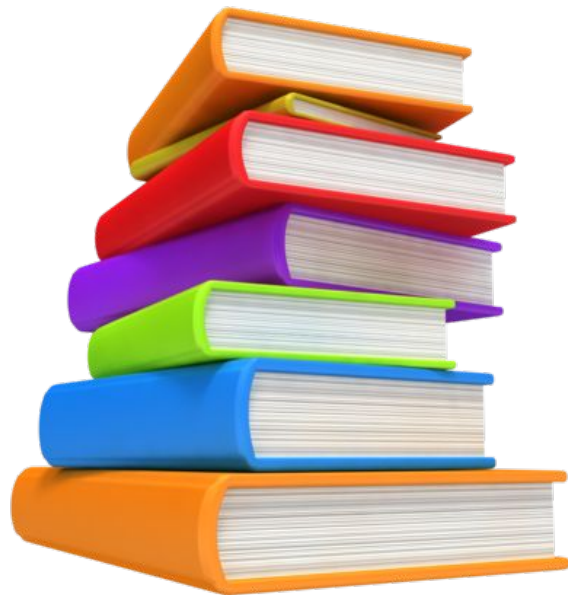
Stack

Sprout



## Stack

- 要怎麼拿到紫色的那本書？



Sprout



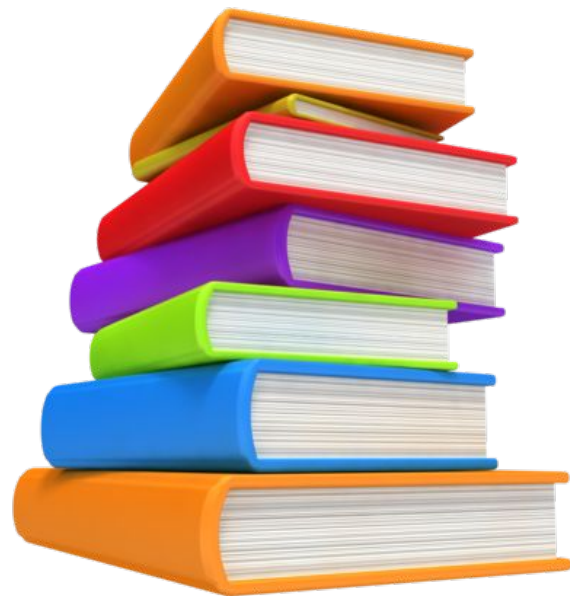


## Stack

- 要怎麼拿到紫色的那本書？

先依序把橘、黃、紅的書拿起來  
拿到紫色的書

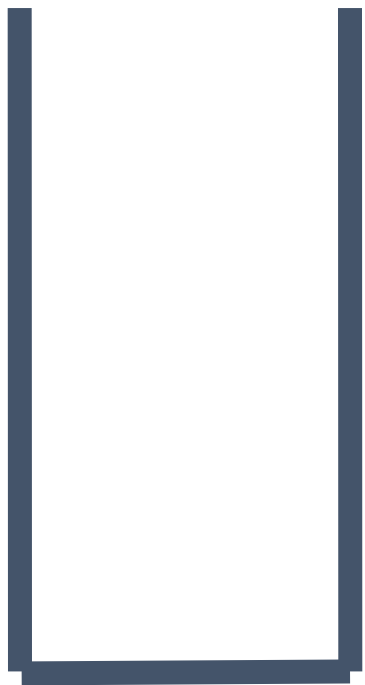
再依序將紅、黃、橘的書放回去



# Sprout



## Stack(堆疊)

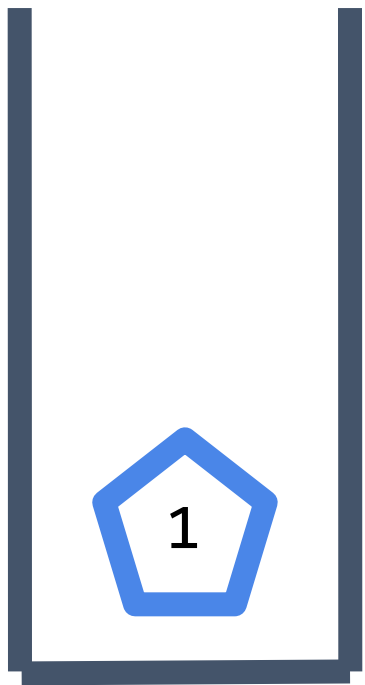


Empty

Sprout



## Stack(堆疊)

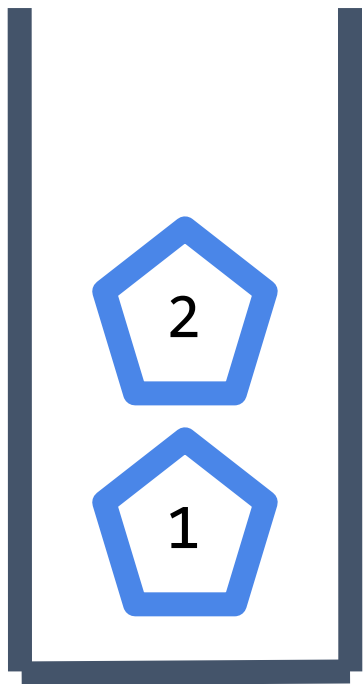


加入新資料

Sprout



## Stack(堆疊)

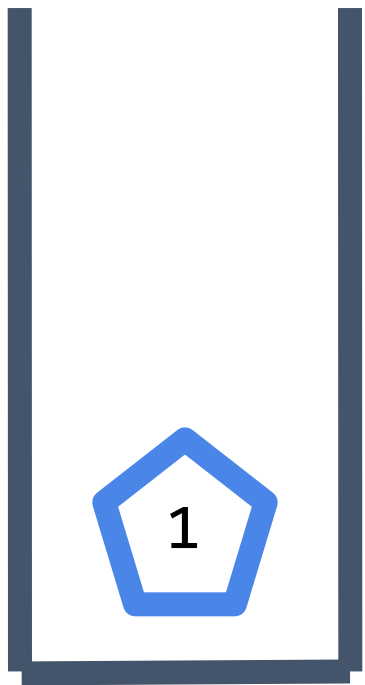


加入新資料

Sprout



## Stack(堆疊)

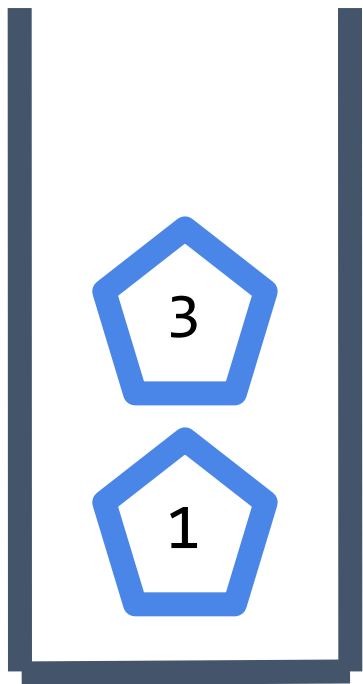


刪除最頂端資料

Sprout



## Stack(堆疊)

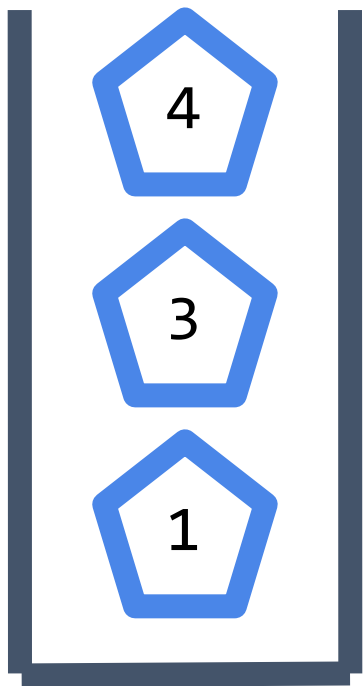


加入新資料

Sprout



## Stack(堆疊)

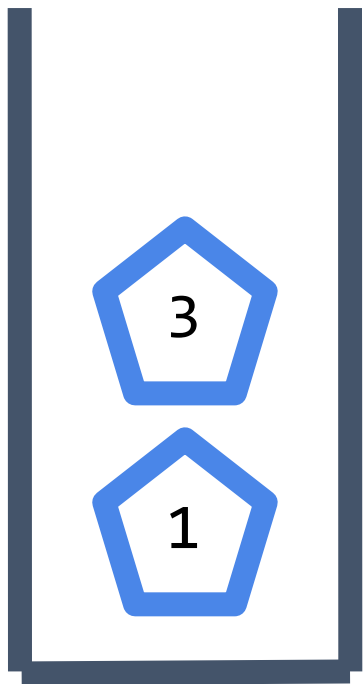


加入新資料

Sprout



## Stack(堆疊)



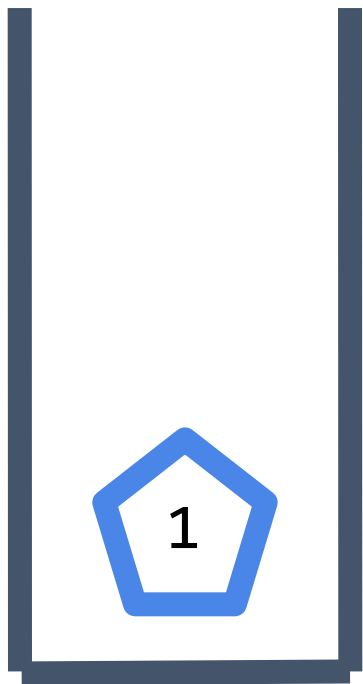
刪除最頂端資料

Sprout





## Stack(堆疊)



刪除最頂端資料

Sprout



## Stack 的功能

- 存取排在 stack 最頂端的資料
- 刪除排在 stack 最頂端的資料
- 新增資料到 stack 的最頂端

Sprout



## Stack 的特性

- 只能從最頂端存取、刪除、新增資料
- 後進先出 (Last In First Out, LIFO)

Sprout



## 用陣列實作 Stack

- `top()` 回傳 stack 最頂端的值
- `pop()` 刪除 stack 最頂端的資料
- `push()` 將一個新的值加入 stack
- `size()` 回傳 stack 的大小

Sprout



## 用陣列代表 Stack

- 假設任意時刻 stack 裡的資料筆數不會超過陣列大小

Sprout



## 用變數 `now` 記錄頂端位置

- 如果 `now = 0`, 代表 `stack` 是空的
- `push` 時 `now++`
- `pop` 時 `now--`

Sprout

```
1  struct Stack{
2      int arr[MAXN], now;
3      Stack() : now(0) {}
4      int top() { //回傳stack最頂端的值
5          return arr[now-1];
6      }
7      void pop() { //刪除stack最頂端的資料
8          now--;
9      }
10     void push(int val) { //將一個新的值加入stack的最頂端
11         arr[now++] = val;
12     }
13     int size() { //回傳stack的大小
14         return now;
15     }
16 };
```



## std::stack

- C++ 的 STL 中, 也有提供 stack 用
- `#include <stack>`
- `std::stack<int> sta;`
- `sta.push(1);`
- `std::cout << sta.top() << '\n';`
- `sta.pop();`

Sprout





## Practice time

- <https://neoj.sprout.tw/problem/36/>
- 寫完的話, 可以挑戰看看 NEOJ #19, #22, #23, #512, #513

Sprout



Queue

Sprout



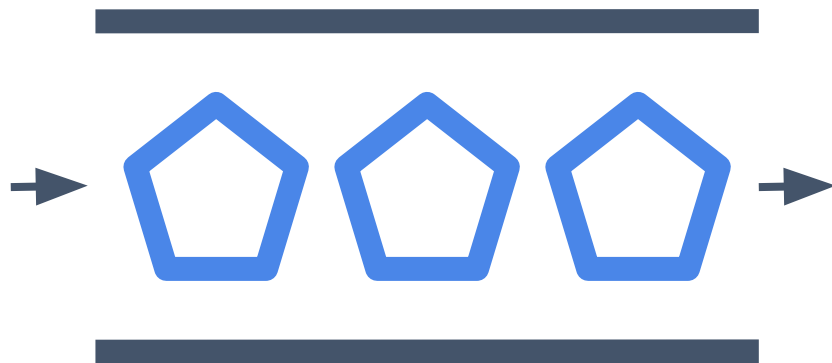
## Queue

- 先到的先拿！





## Queue (佇列)



Sprout



## Queue 的功能

- 存取排在 queue 最前端的資料
- 刪除排在 queue 最前端的資料
- 新增資料到 queue 的最後端

Sprout



## Queue 的特性

- 只能從最前端存取、刪除資料
- 只能從最後端新增資料
- 先進先出(First In First Out, FIFO)

Sprout



## 用陣列實作 Queue

- `front()` 回傳 queue 最前端的值
- `pop()` 刪除 queue 最前端的資料
- `push()` 將一個新的值加入 queue 的最後端
- `size()` 回傳 queue 的大小

Sprout



## 和 stack 類似的方法

- 用變數 head, tail 記錄目前 queue 的開頭、結尾
- push 時 `tail++`
- pop 時 `head++`

Sprout





## 但...可能會碰到問題

- 如果一直 push 東西進去後立刻 pop 出來？

Sprout



## 但...可能會碰到問題

- 如果一直 push 東西進去後立刻 pop 出來？
- 雖然在過程中 queue 所存的東西數量不會超過上限，但？
- 你會發現你的 head 跟 tail 數字越來越大
- 索引值有可能出界

Sprout



## Circular Queue

- 如果不斷進行 push 然後 pop 的操作會超過陣列限制

Sprout



## Circular Queue

- 如果不斷進行 push 然後 pop 的操作會超過陣列限制
- 碰到尾巴的話, 就從頭再來一次 !

Sprout

```
1 struct Queue{
2     int arr[MAXN], head, tail;
3     Queue() : head(0), tail(0) {}
4     int front() { //回傳queue最前端的值
5         return arr[head];
6     }
7     void pop() { //刪除queue最前端的資料
8         head++;
9         if (head == MAXN) head = 0;
10    }
11    void push(int val) { //將一個新的值加入queue的最後端
12        arr[tail++] = val;
13        if (tail == MAXN) tail = 0;
14    }
15    int size() { //回傳queue的大小
16        return (tail + MAXN - head) % MAXN;
17    }
18 };
```



## std::queue

- C++ 的 STL 中, 也有提供 queue 用
- `#include <queue>`
- `std::queue<int> que;`
- `que.push(1);`
- `std::cout << que.front() << '\n';`
- `que.pop();`

Sprout



## Practice time

- <https://neoj.sprout.tw/problem/37/>
- 寫完的話, 可以挑戰看看 NEOJ #20
- 特別注意到宣告一個 `std::queue` 就會佔用一些記憶體了, 再寫 NEOJ #20 的時候, 注意不要開  $10^5$  量級個 `queue` (?)
  - source:  
[https://gcc.gnu.org/bugzilla/show\\_bug.cgi?id=77524](https://gcc.gnu.org/bugzilla/show_bug.cgi?id=77524)

Sprout



Deque

Sprout





## Deque (雙端佇列)

- 有些人喜歡念成「de-queue」

Sprout



## Deque (雙端佇列)

- ~~有些人喜歡念成「de-queue」~~
- usually pronounced like "deck" — by C++ reference

Sprout



## Deque 的功能

- 存取、刪除排在 deque 最前端的資料
- 存取、刪除排在 deque 最後端的資料
- 新增資料到 deque 的最前端、最後端

Sprout



## 用陣列實作 Deque

- `front()`, `back()` 詢問
- `pop_front()`, `pop_back()` 刪除
- `push_front()`, `push_back()` 加入
- `size()`

Sprout



```
1 struct Deque{
2     int arr[MAXN], head, tail;
3     Deque() : head(0), tail(0) {}
4     int front() { //回傳deque最前端的值
5         return arr[head];
6     }
7     int back() { //回傳deque最後端的值
8         if (tail == 0) tail = MAXN
9         return arr[tail-1];
10    }
11    void pop_front() { //刪除deque最前端的資料
12        head++;
13        if (head == MAXN) head = 0;
14    }
15    void pop_back() { //刪除deque最後端的資料
16        if (tail == 0) tail = MAXN;
17        tail--;
18    }
19    void push_front(int val) { //將一個新的值加入deque的最前端
20        if (head == 0) head = MAXN-1;
21        arr[head--] = val;
22    }
23    void push_back(int val) { //將一個新的值加入deque的最後端
24        arr[tail++] = val;
25        if (tail == MAXN) tail = 0;
26    }
27    int size() { //回傳deque的大小
28        return (tail + MAXN - head) % MAXN;
29    }
30 };
```

out



## std::deque

- C++ 的 STL 中, 也有提供 deque 用
- `#include <queue>`
- `std::deque<int> dq;`
- `dq.push_front(1);`
- `dq.push_back(2);`
- `dq[0] = 3;`
- `dq.pop_front();`
- `dq.pop_back();`

Sprout



## 例題討論

Sprout



## 例題一、括弧匹配

Sprout





## 題目敘述

給定一個僅包含 '('、')' 的字串，問其是否為合法括弧字串。

範例：

"()()()()" 是一個合法括弧字串

"()((()())" 不是一個合法括弧字串

Sprout



## Hint

- 什麼樣的字串是合法括弧字串？

Sprout



## Hint

- 什麼樣的字串是合法括弧字串？
  - 「每個左括弧都能夠找到右括弧與其互相配對，且不會有多餘的右括弧沒有配對到」

Sprout



## 作法

- 由左到右把字元加到 stack 看看
  - 遇到 '(' 就 push
  - 遇到 ')' 就 pop
- 什麼樣的情況是非法字串？

Sprout



## 作法

- 由左到右把字元加到 stack 看看
  - 遇到 '(' 就 push
  - 遇到 ')' 就 pop
- 什麼樣的情況是非法字串？
  - 如果 pop 的時候發現 stack 空了 => 非法字串
  - 如果 stack 最後不是空的 => 非法字串

Sprout



## 例題二、加減運算

Sprout



## 題目敘述

給定一個包含 '('、')'、'+'、'-' 的運算式，計算該運算式的答案。（保證該運算式合法）

範例：

"(5+4)-3" 的答案是 6

"(1+2)-(7+3)" 的答案是 -7

Sprout



## Hint

- 可以從後面做回來
- 兩個 Stack 比較好實作

Sprout





## 作法

- 兩個 stack, 一個紀錄符號、一個紀錄數值
- 碰到 '(' 或結尾再做事！

--	--	--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--	--	--

$$11 - ( 2 - ( 3 + 2 ) ) + 5$$

Sprout



## 還有一個小問題

- 數字不是個位數怎麼辦？

Sprout



## 實作細節

- 開一個變數紀錄目前的 `digit` 是  $10$  的幾次方, 先處理好數字再丟進 `stack` 中
- 遇到 `'('` 往前計算的時候不是只算一次, 是一路到 `')'` 並將其 `pop` 出來為止!

Sprout



## 延伸問題

給定一個包含加減乘除和括弧的四則運算式，計算其答案。

Sprout



### 例題三、長條圖最大矩形

Sprout

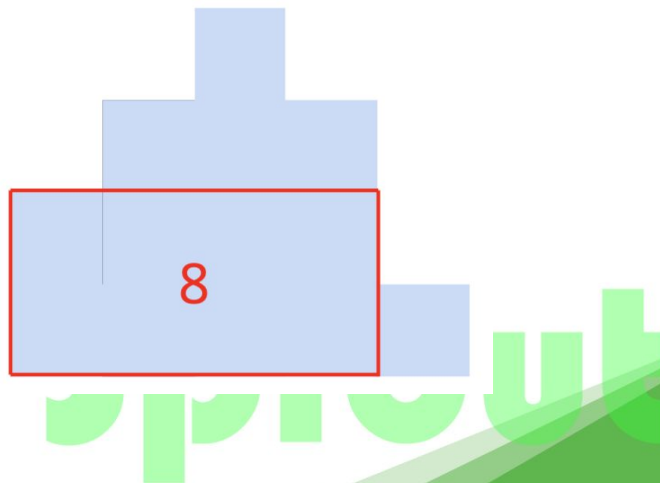
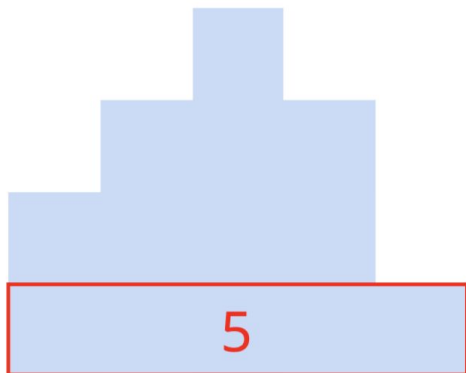


## 題目敘述

給你一張長條圖每個位置的高度，問你能畫出的最大矩形面積。（ $N \leq 10^5$ 、高度  $\leq 10^9$ ）

範例：

2 3 4 3 1



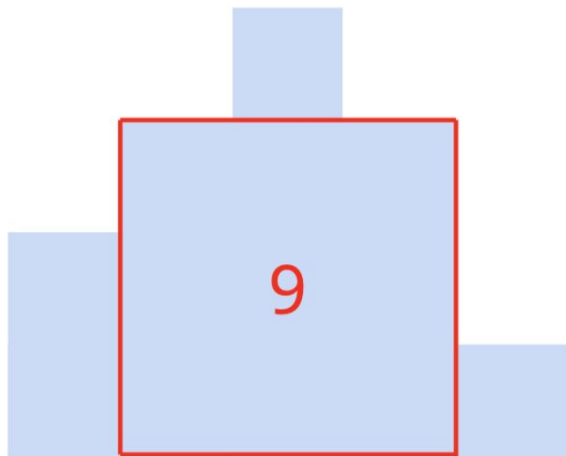


## 題目敘述

給你一張長條圖每個位置的高度，問你能畫出的最大矩形面積。（ $N \leq 10^5$ 、高度  $\leq 10^9$ ）

範例：

2 3 4 3 1



prout



## Hint

- 不知道從何下手的時候，可以先從複雜度較差的解開始想！

Sprout





## 直覺的做法

- 枚舉每段區間，然後看高度最高可以是多少
- 正確性？
- 複雜度？ $O(N^3)$ 
  - 區間總共有  $N(N+1)/2$  個
  - 高度至多只能到最矮的那個  $\Rightarrow$  掃一遍區間找最小值

Sprout



再想多一點...

- 「一塊區間的高度至多只能到最矮的那個」

Sprout



## 再想多一點...

- 「一塊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

Sprout



## 再想多一點...

- 「一塊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？
  - 只要分別找到左右兩邊第一個比我小的！

Sprout



## 問題轉換

- 給定序列，對每一項分別找到左右離他最近且比他小的值。  
( $N \leq 10^5$ 、值域  $\leq 10^9$  )
  - 其實等價於對每一項找到左邊離他最近且比他小的值，然後再把序列反轉過來做一次
  - 複雜度？
    - 但有沒有可能做得更好呢？

Sprout



## 作法

- 考慮每一項在什麼時間點以後注定不可能成為答案

Sprout



## 作法

- 考慮每一項在什麼時間點以後注定不可能成為答案
- 「如果右邊有東西不比我大，那我就不可能是答案」
  - 我們要用 stack 維護這樣的「單調性」
  - stack 裡頭的每一項一定比前一項大

Sprout



## 作法

- 考慮每一項在什麼時間點以後注定不可能成為答案
- 「如果右邊有東西不比我大，那我就不可能是答案」
  - 我們要用 stack 維護這樣的「單調性」
  - stack 裡頭的每一項一定比前一項大

```
1 while (size() > 0 && top() >= value[idx]) { // top比我還大
2     pop(); // 把top丟掉
3 }
4 if (size() > 0) ans[idx] = top();
5 // 目前的top會是比我小且離我最近的那個
6 push(value[idx]); // 記得把值丟進stack
```





## 思路、步驟整理

1. 要找最大矩形，可以「枚舉每個值作為最小值」向外延伸
2. 將向左、向右拆開成兩個問題
3. 題目轉化為「找到左邊第一個比我小的值」
4. 一個值不可能成為最小值的條件（右邊出現比它小的值）
5. 利用 `stack` 維護這樣的「單調遞增」

Sprout



## 延伸問題

給定長度為  $N$  的序列，問每個長度  $K$  連續區間的區間最大值。  
( $N, K \leq 10^6$ )

Sprout



Linked-list

Sprout



## Linked-list 的概念

- 對於每個資料紀錄前後資料的位置
- 可以  $O(1)$  加入、刪除特定資料
- 不支援 random-access
  - 不能  $O(1)$  存取指定 index 的資料

Sprout



## Linked-list 的概念

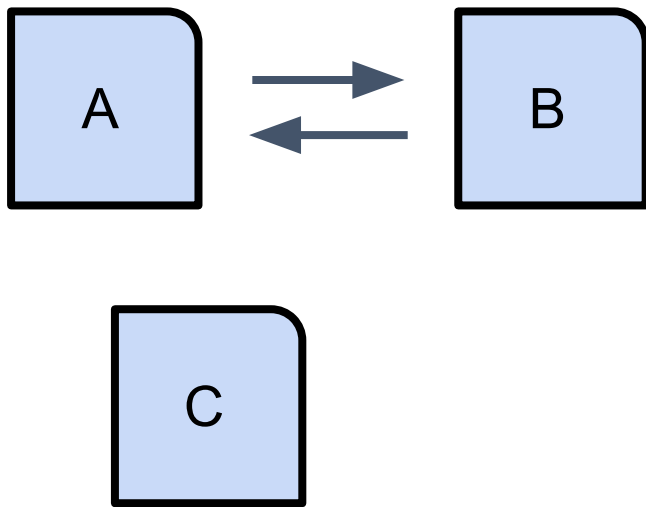
- 對於每個資料紀錄前後資料的位置
- 可以  $O(1)$  加入、刪除特定資料
- 不支援 random-access
  - 不能  $O(1)$  存取指定 index 的資料
- 這跟陣列不一樣的地方在哪？

Sprout



## 加入資料

- 假設我們想將資料 C 插入在資料 A、B 之間

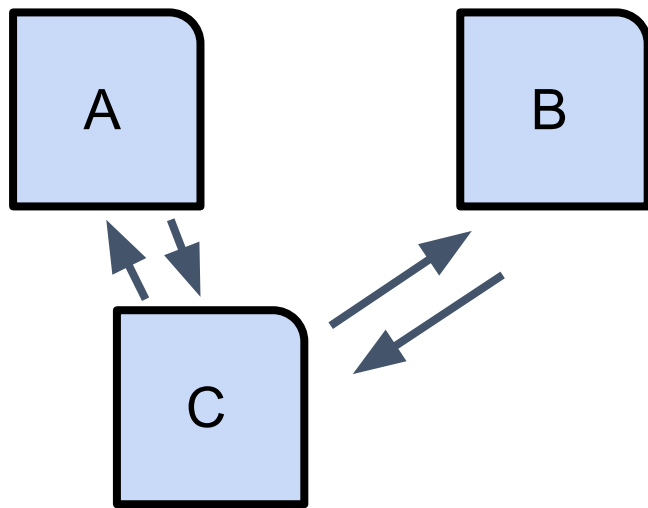


Sprout



## 加入資料

- 改變他們指向前後的那些箭頭！

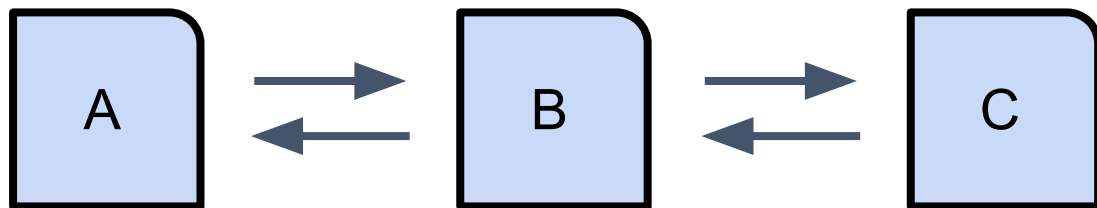


Sprout



## 刪除資料

- 假設我們想將資料 B 從資料 A、C 之間刪除



Sprout





## 實作

- 實做

- 先定義一個struct/class Node, 作為linked list的節點, 裡面存資訊和一個指向下一個Node的指標
- 使用時只用一個變數head記錄linked list的起點就可以了

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int _data;
    Node* _next;
}

int main () {
    Node* node = new Node();
}
```

Sprout



## 另外一種作法

- 使用陣列的 index

```
struct Node {  
    int _data;  
    int _next;  
} node[1000006];
```

```
int main () {  
    Node head;  
    head._next = 1;  
}
```

Sprout



## Try it

- Try NEOJ #21, #25
- #21 Hint: <https://pastebin.com/sWRRxrpz>

Sprout



謝謝大家！

Sprout