Data Structure

Lecture by casperwang
2022/03/05

Sprout

課程內容

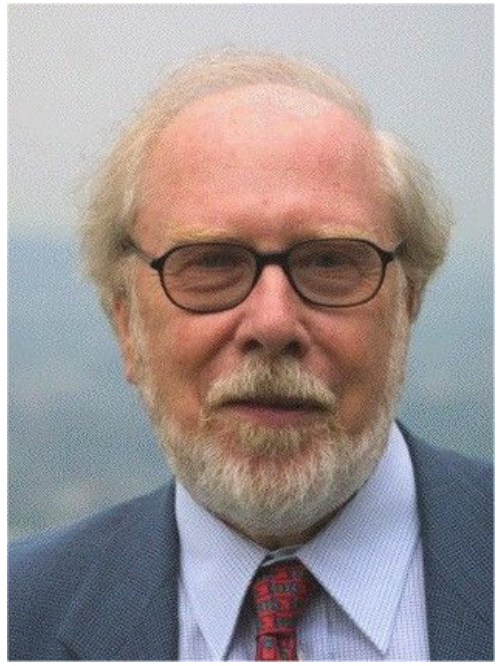
- 什麼是資料結構？
- Stack
- Queue
- Deque
- Linked-list
- 例題討論

什麼是資料結構？

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

Why 資料結構？



Algorithms + Data Structures
= Programs

Niklaus Wirth,
the designer of Pascal

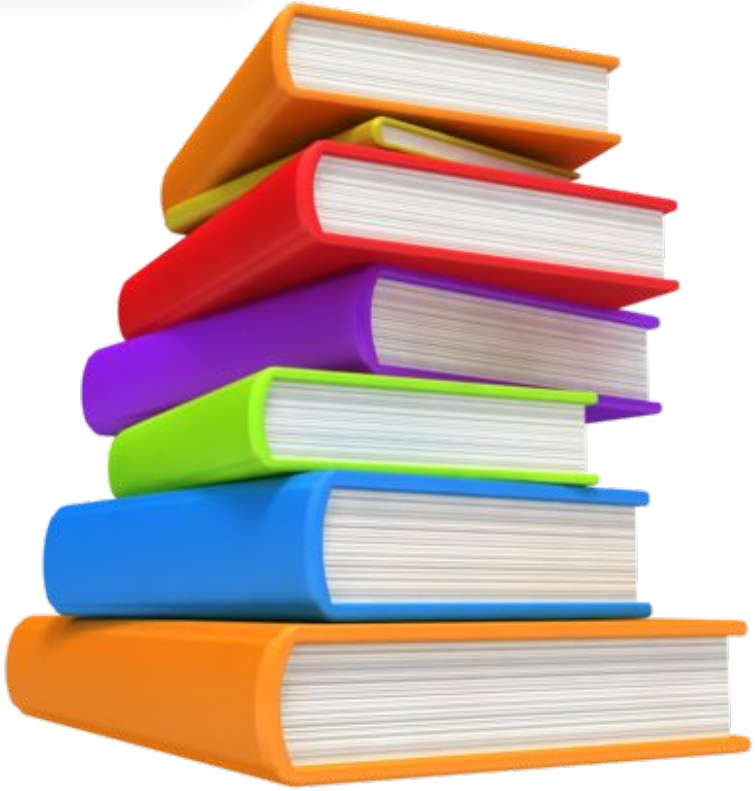
常見的資料結構

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

Stack

Stack

- 要怎麼拿到紫色的那本書？



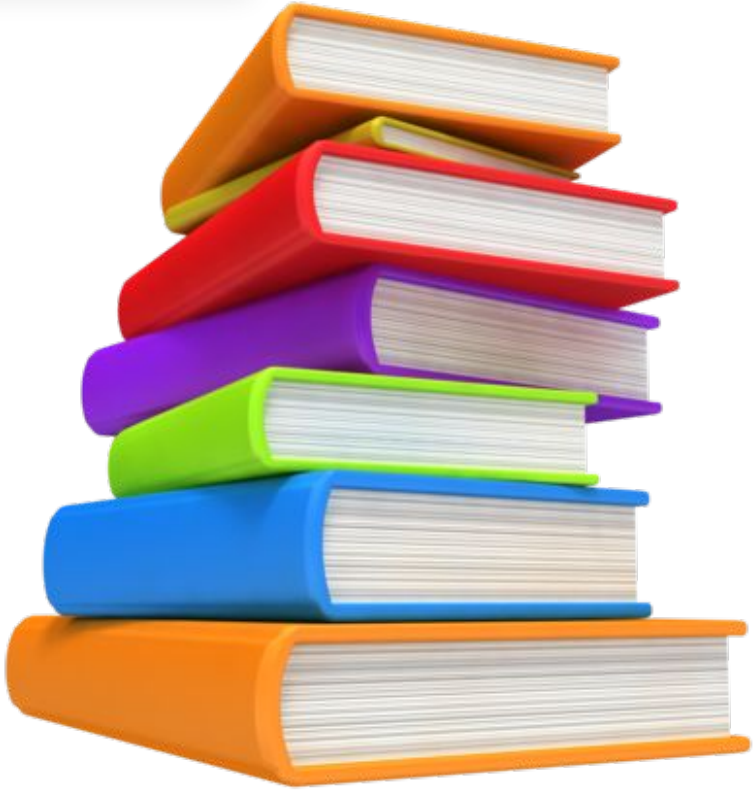
Stack

- 要怎麼拿到紫色的那本書？

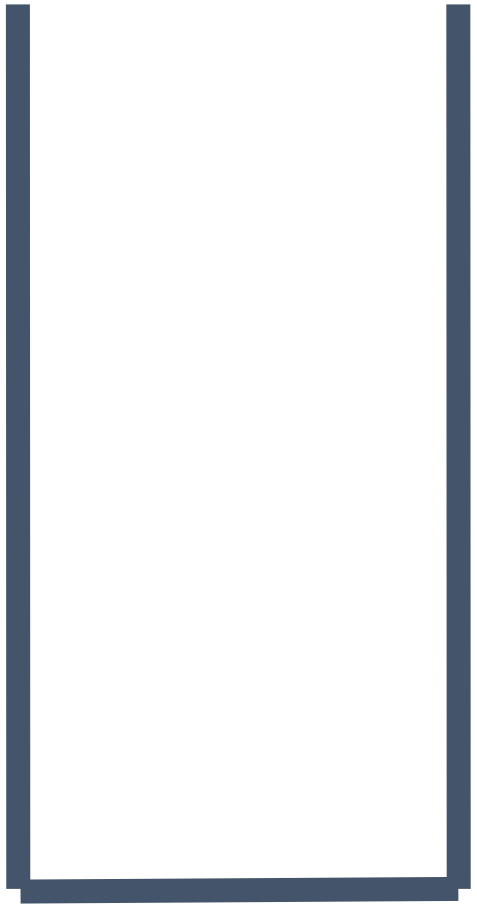
先依序把橘、黃、紅的書拿起來

拿到紫色的書

再依序將紅、黃、橘的書放回去

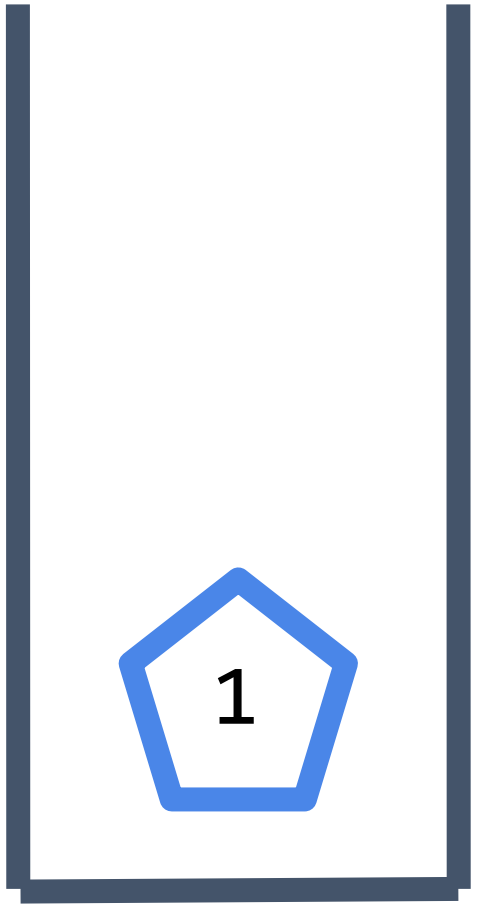


Stack(堆疊)



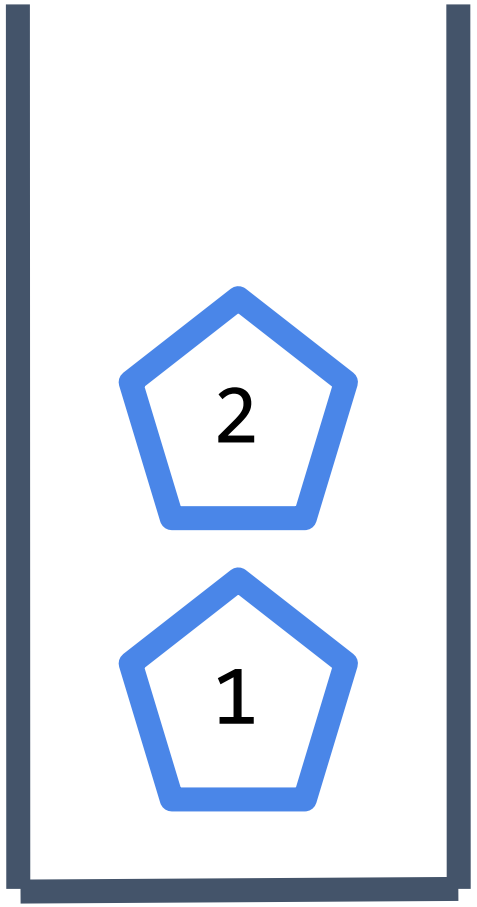
Empty

Stack(堆疊)



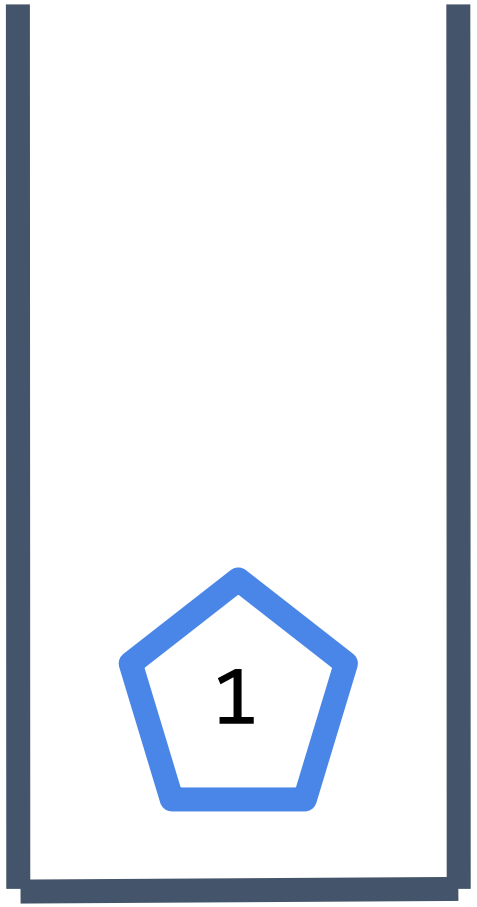
加入新資料

Stack(堆疊)



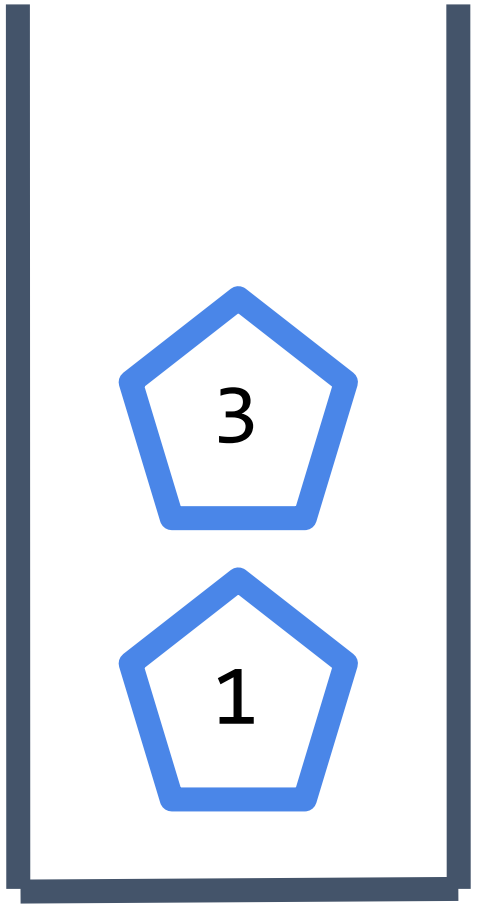
加入新資料

Stack(堆疊)



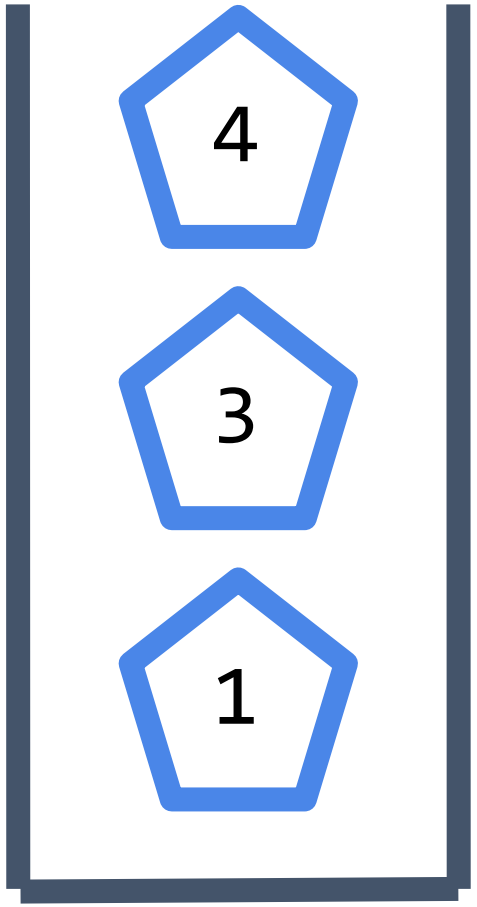
刪除最頂端資料

Stack(堆疊)



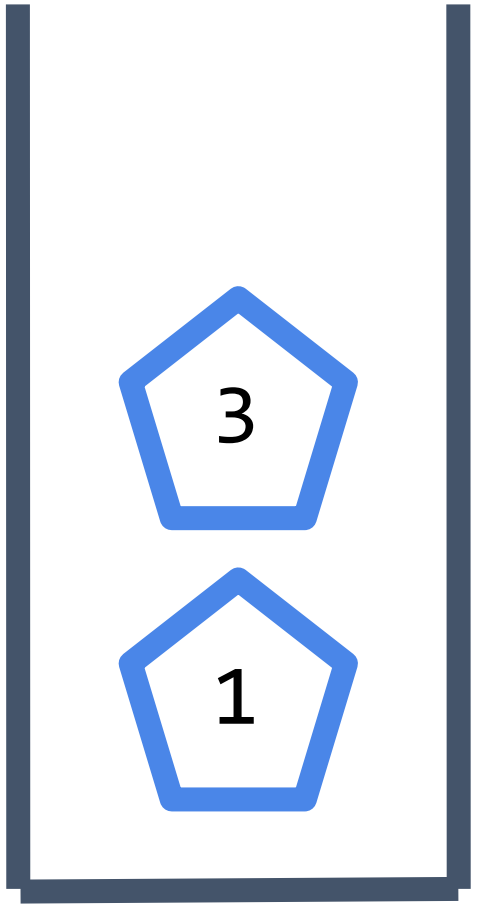
加入新資料

Stack(堆疊)



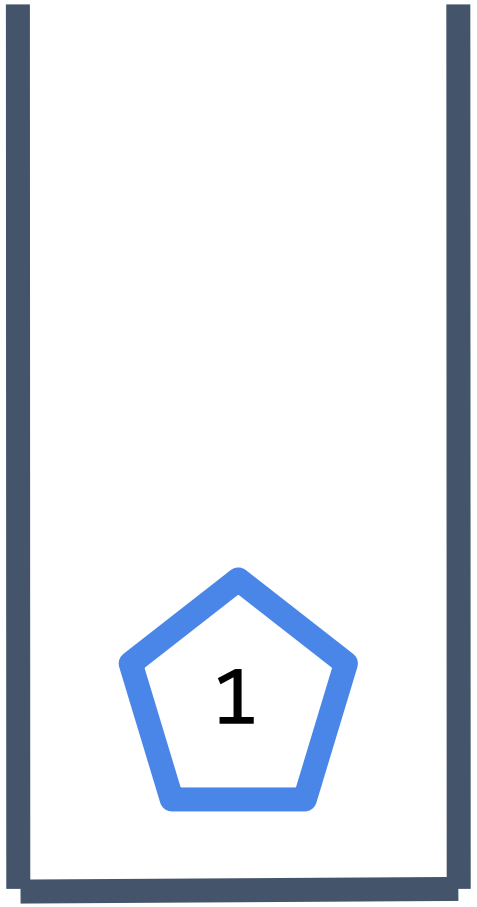
加入新資料

Stack(堆疊)



刪除最頂端資料

Stack(堆疊)



刪除最頂端資料

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

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)

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

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

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--

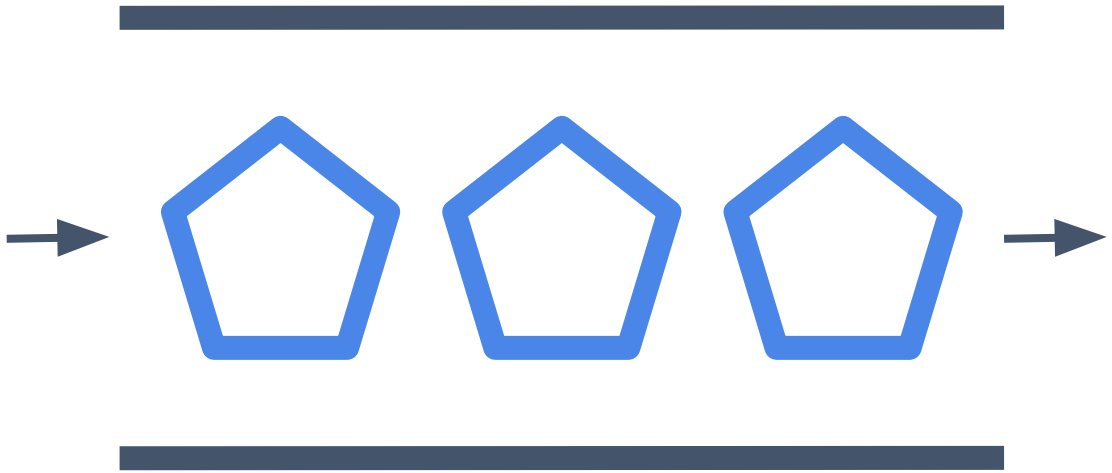
```
1  struct Stack{
2      int arr[MAXN], now;
3      Stack() : now(0) {}
4      int top() { //回傳stack最頂端的值
5          return arr[now-1];
6      }
7      void pop() { //刪除stack最頂端的資料
8          now--;
9      }
10     void push(int val) { //將一個新的值加入stack的最頂端
11         arr[now++] = val;
12     }
13 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

Queue

Queue

- 先到的先拿！

Queue（佇列）



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

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)

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

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++

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++

但...可能會碰到問題

- 如果一直 push 東西進去後立刻 pop 出來？

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？

Circular Queue

- 如果不斷進行 push 然後 pop 的操作會超過陣列限制

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！

```
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 };
```

Deque

Deque（雙端佇列）

- 有些人喜歡念成「de-queue」

Deque（雙端佇列）

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

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

用陣列實作 Deque

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


```
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 };
```

Linked-list

Linked-list 的概念

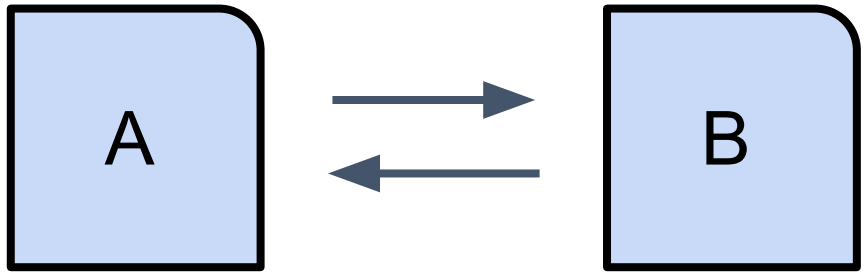
- 對於每個資料紀錄前後資料的位置
- 可以 O(1) 加入、刪除特定資料
- 不支援 random-access
  - 不能 O(1) 存取指定 index 的資料

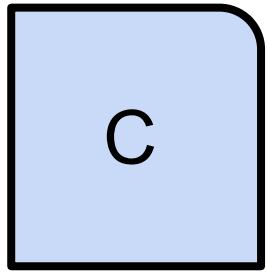
Linked-list 的概念

- 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？

加入資料

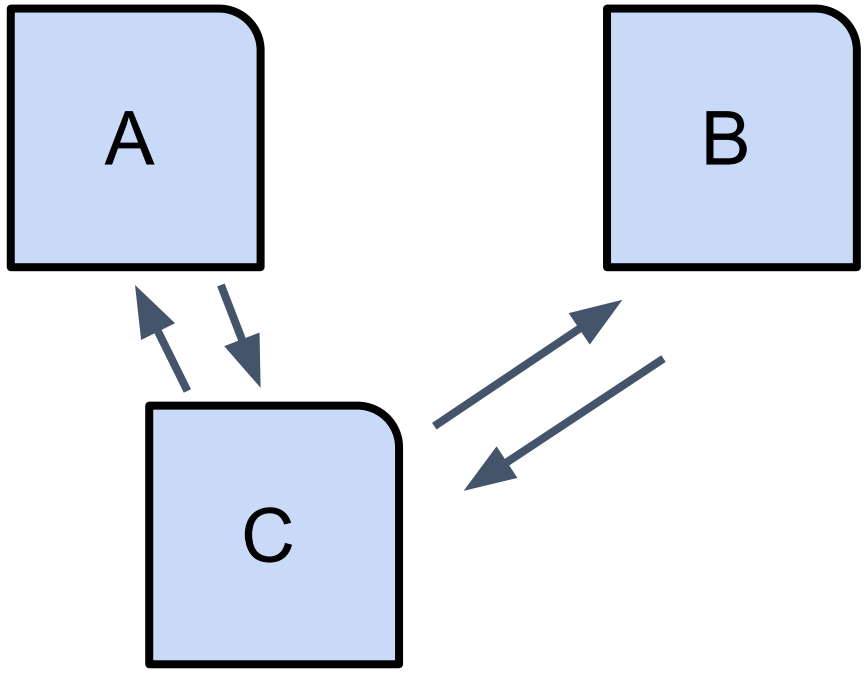
- 假設我們想將資料 C 插入在資料 A、B 之間





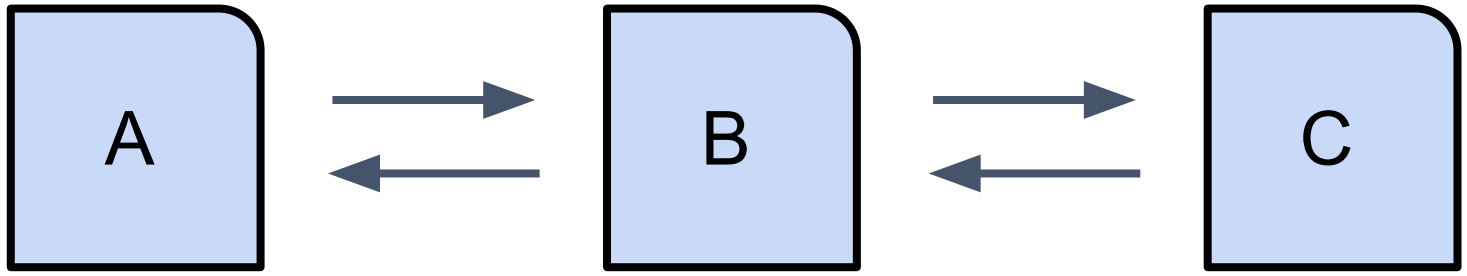
加入資料

- 改變他們指向前後的那些箭頭！



刪除資料

- 假設我們想將資料 B 從資料 A、C 之間刪除



例題討論

括弧匹配

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

括弧匹配

- 什麼樣的字串是合法括弧字串？

括弧匹配

- 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」

括弧匹配

- 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？

括弧匹配

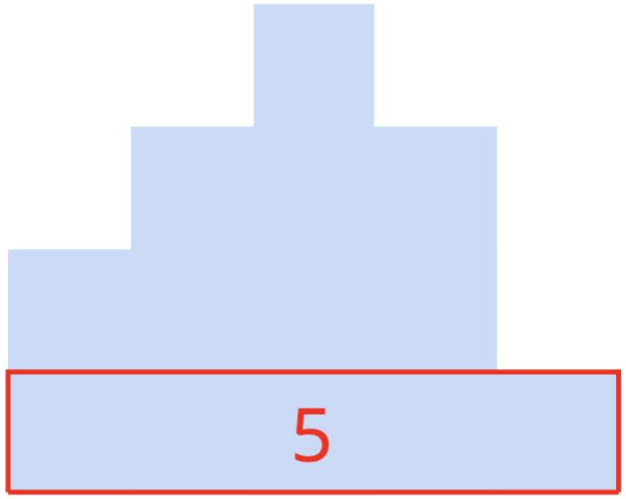
- 由左到右把字元加到 stack 看看
  - 遇到 '(' 就 push
  - 遇到 ')' 就 pop
- 什麼樣的情況是非法字串？
  - 如果 pop 的時候發現 stack 空了 => 非法字串
  - 如果 stack 最後不是空的 => 非法字串

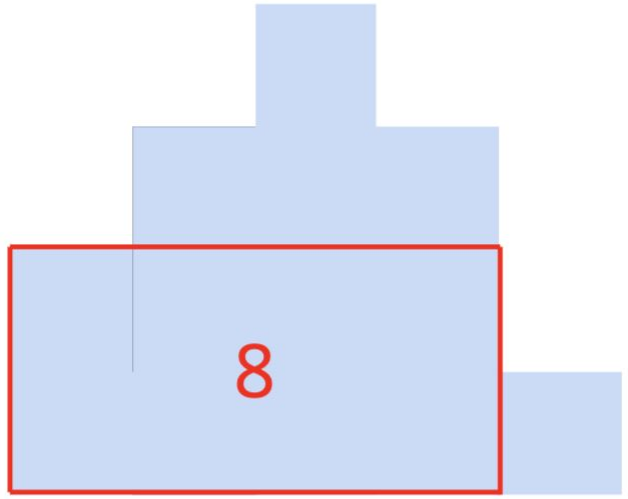
長條圖最大矩形

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 <= 10^5、高度 <= 10^9）

範例：

2 3 4 3 1



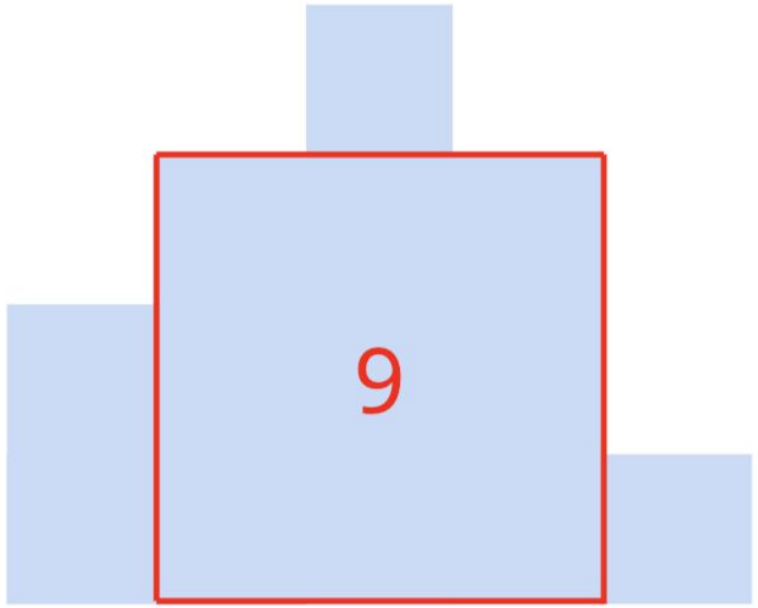


長條圖最大矩形

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 <= 10^5、高度 <= 10^9）

範例：

2 3 4 3 1



長條圖最大矩形

- 不知道從何下手的時候，可以先從複雜度較差的解開始想！

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 => 掃一遍區間找最小值

再想多一點...

- 「一塊區間的高度至多只能到最矮的那個」

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

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！

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 <= 10^5、值域 <= 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？

作法

- 考慮每一項在什麼時間點以後注定不可能成為答案

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

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」

謝謝大家！

Sprout