Data Structures

Lecture by pudding164253

2025.02.22 Algorithm-Tainan

課程內容

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

Sprout

什麼是資料結構？

資料結構

- 一種儲存資料的方式
- 好的儲存方式能更快的找到想要的東西
- 或是用更小的空間存好資料
- 舉例來說像是好好分類的資料夾

常見的資料結構

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

Sprout

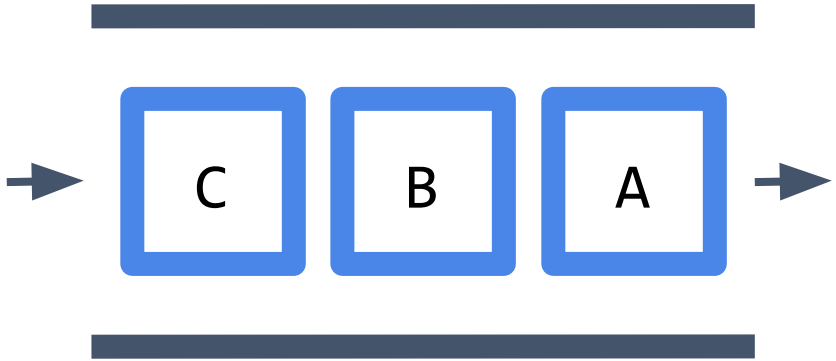
Queue

Queue

- 先到的先拿！

Queue

- 現在有一堆小方塊在排隊



Queue 的功能與特性

- 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
- 先進先出(First In First Out, FIFO)

用陣列實作 Queue

- 存取排在 queue 最前端的資料 -> front()
- 刪除排在 queue 最前端的資料 -> pop()
- 新增資料到 queue 的最後端 -> push()
- 得到 queue 裡面有多少資料 -> size()
- 知道 queue 現在是不是空的 -> empty()
- 後面的操作是比照 STL 格式
- 可是 STL 是啥？為甚麼要這樣設計

STL

- STL(Standard Template Library) 標準模板庫
- 把一些實用的東西都寫好了
- 有的東西有被優化，但也有一些安全性考量所以不一定更快

std::queue

- STL 的 queue 是這樣用的

- #include <queue>

- std::queue<int> que;
- que.push(1);
- std::cout << que.front() << que.size() << ‘\n’;
- if(!que.empty())que.pop();

陣列實作 Queue

- 那如果要自己實作呢
- 開一段陣列儲存，push 時就把東西加在最後面
- 順便紀錄最後一項的位置，empty 判斷位置是不是 0 就好
- front 直接拿第一項，size 也是直接回傳長度
- pop 的時候把東西都往前移，感覺太慢了
- 那假裝前面的值不存在，紀錄真正的開頭位置

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？
- 頭尾會一直往後移動，有可能移出界
- 那就循環的利用前面不要的空間！

```
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 };
```

Practice time

- https://tioj.sprout.tw/problems/36/
- 寫完的話，可以挑戰看看 Sprout OJ #19
- 注意到宣告一個 std::queue 就會佔用一些記憶體了，在寫 Sprout OJ #19 的時候，注意不要開 1e5 個 queue
  - source: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=77524

Deque

Deque(雙向的 Queue)

- 有人念 de-queue
- 有人念 /dɛk/
- 反正我是念 deque

Deque 的功能與特性

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

用陣列實作 Deque

- 存取排在 deque 最前端和最後端的資料 -> front(), back()
- 刪除排在 deque 最前端和最後端的資料 -> pop_front(), pop_back()
- 新增資料到 deque 的最後端和最前端 -> push_front(), push_back()
- 得到 deque 裡面有多少資料 -> size()
- 知道 deque 現在是不是空的 -> empty()

```cpp
struct Deque{
    int arr[MAXN], head, tail;
    Deque() : head(0), tail(0) {}
    int front() { //回傳deque最前端的值
        return arr[head];
    }
    int back() { //回傳deque最後端的值
        if (tail == 0) tail = MAXN
        return arr[tail-1];
    }
    void pop_front() { //刪除deque最前端的資料
        head++;
        if (head == MAXN) head = 0;
    }
    void pop_back() { //刪除deque最後端的資料
        if (tail == 0) tail = MAXN;
        tail--;
    }
    void push_front(int val) { //將一個新的值加入deque的最前端
        if (head == 0) head = MAXN-1;
        arr[head--] = val;
    }
    void push_back(int val) { //將一個新的值加入deque的最後端
        arr[tail++] = val;
        if (tail == MAXN) tail = 0;
    }
    int size() { //回傳deque的大小
        return (tail + MAXN - head) % MAXN;
    }
};
```

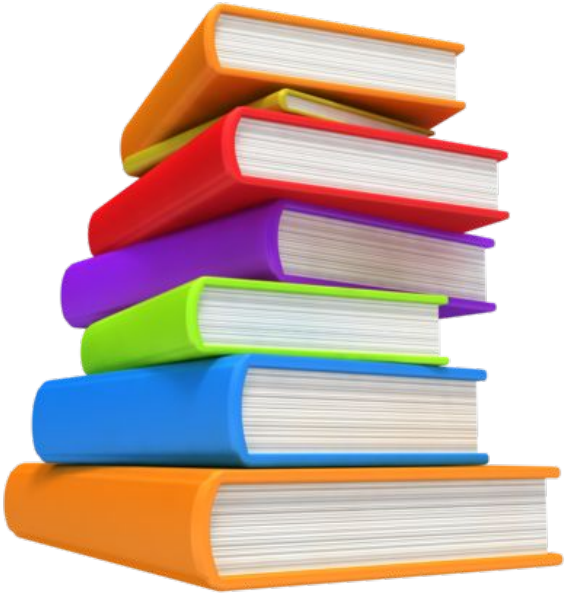
std::deque

- STL 也有提供 deque
- #include <queue> //一樣在 queue 裡面
- std::deque<int> deq;
- deq.push_front(1);
- deq.push_back(2);
- deq.pop_front();
- deq.pop_back();
- std::cout << deq.front() << deq.size() << '\n';

Stack

Stack

- 要怎麼拿到紫色的那本書？



Sprout

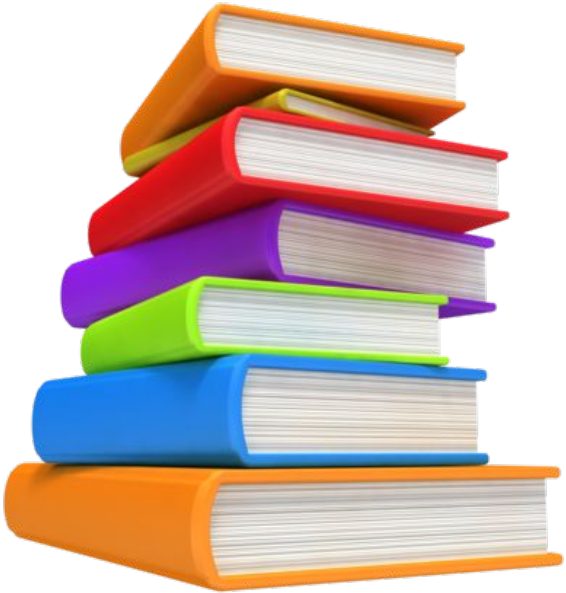
Stack

- 要怎麼拿到紫色的那本書？

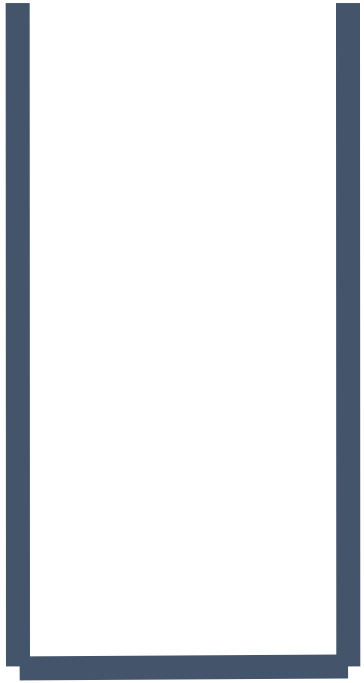
先依序把橘、黃、紅的書拿起來

拿到紫色的書

再依序將紅、黃、橘的書放回去



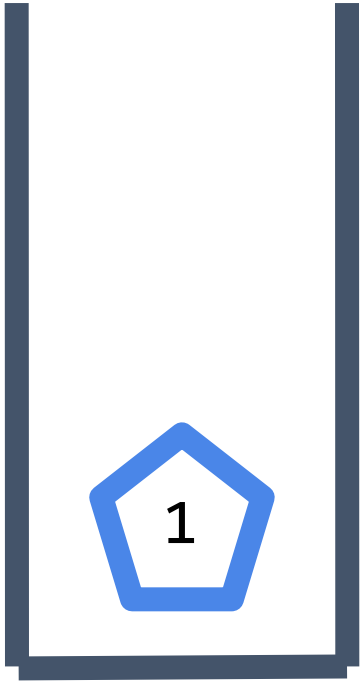
Stack(堆疊)



Empty

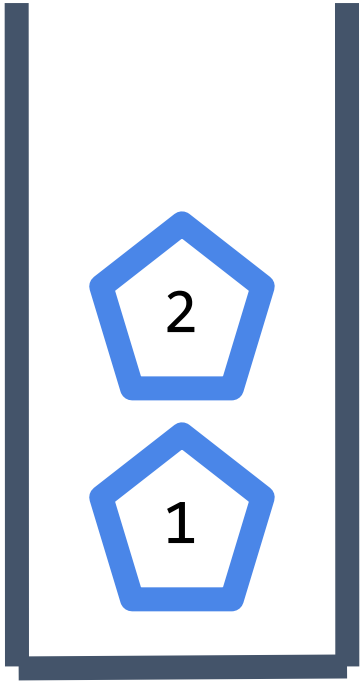
Sprout

Stack(堆疊)



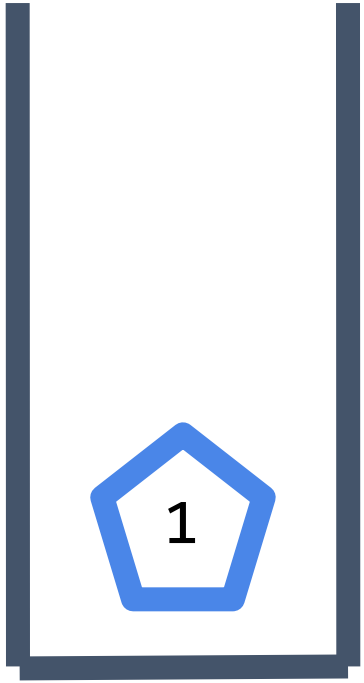
加入新資料

Stack(堆疊)



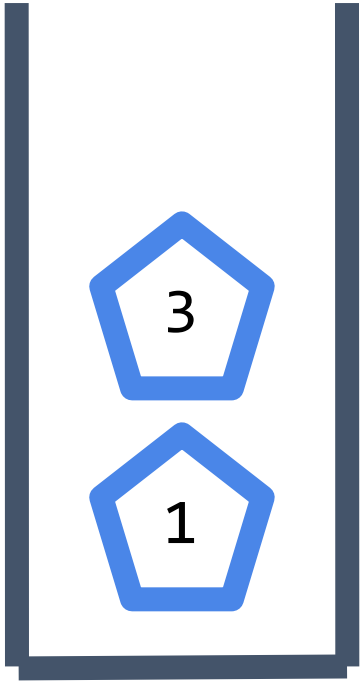
加入新資料

Stack(堆疊)



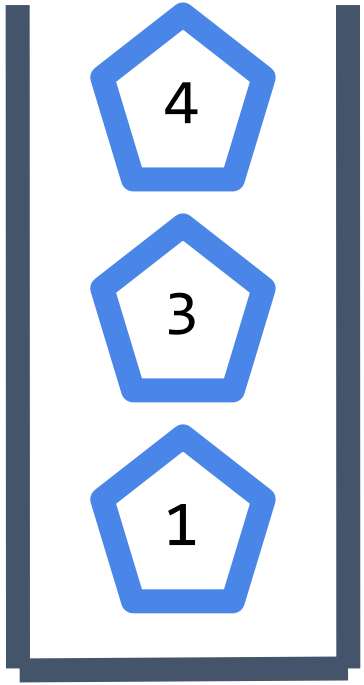
刪除最頂端資料

Stack(堆疊)



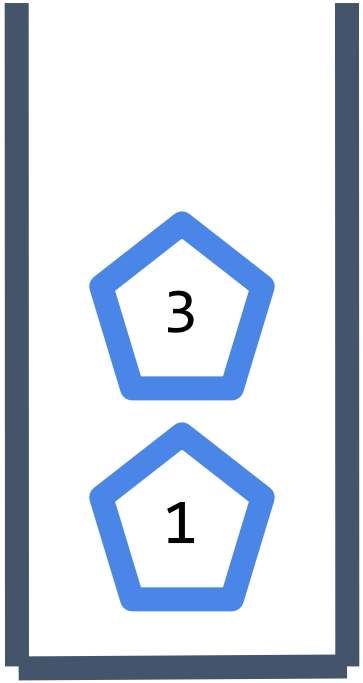
加入新資料

Stack(堆疊)



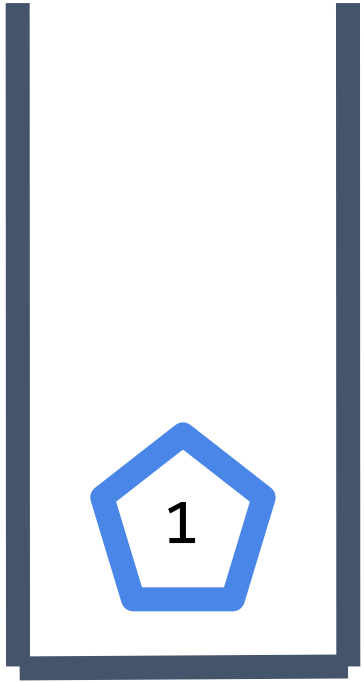
加入新資料

Stack(堆疊)



刪除最頂端資料

Stack(堆疊)



刪除最頂端資料

Stack 的功能與特性

- 存取排在 stack 最上端的資料
- 刪除排在 stack 最上端的資料
- 新增資料到 stack 的最上端
- 先進後出(First In Last Out, FILO)
- 所有操作都只能在最上端

用陣列實作 Stack

- 存取排在 stack 最上端的資料 -> top()
- 刪除排在 stack 最上端的資料 -> pop()
- 新增資料到 stack 的最上端 -> push()
- 得到 stack 裡面有多少資料 -> size()
- 知道 stack 現在是不是空的 -> empty()
- 陣列大小需要至少是 stack 裡的東西最大數量

```cpp
struct Stack{
    int arr[MAXN], now;
    Stack() : now(0) {}
    int top() { //回傳stack最頂端的值
        return arr[now-1];
    }
    void pop() { //刪除stack最頂端的資料
        now--;
    }
    void push(int val) { //將一個新的值加入stack的最頂端
        arr[now++] = val;
    }
    int size() { //回傳stack的大小
        return now;
    }
};
```

std::stack

- 理所當然的, STL 中也還是有 stack

- #include <stack>

- std::stack<int> sta;
- sta.push(1);
- std::cout << sta.top() << ‘\n’;
- sta.pop();

Practice time

- https://tioj.sprout.tw/problems/35/
- 寫完的話，可以挑戰看看 Sprout OJ #18, #21, #22, #424, #425

例題討論

例題一、括弧匹配

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

Hint

- 什麼樣的字串是合法括弧字串？

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」

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？

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

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
- 實際上不用真的存一個 stack，只要記錄有幾個左括號就好

例題二、加減運算

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

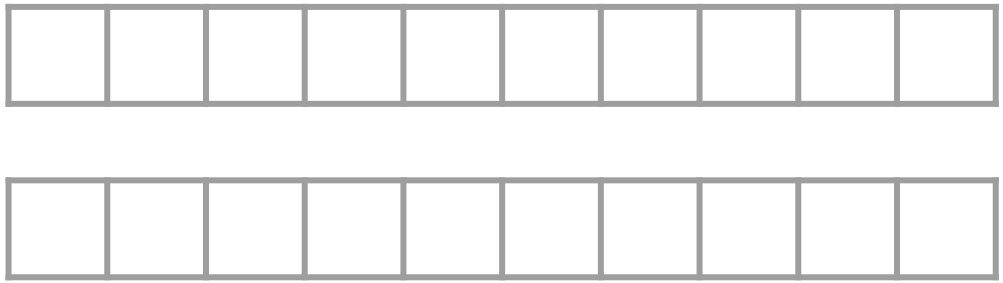
"(1+2)-(7+3)" 的答案是 -7

Hint

- 可以從後面做回來
- 兩個 Stack 比較好實作

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

還有一個小問題

- 數字不是個位數怎麼辦？

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！

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。

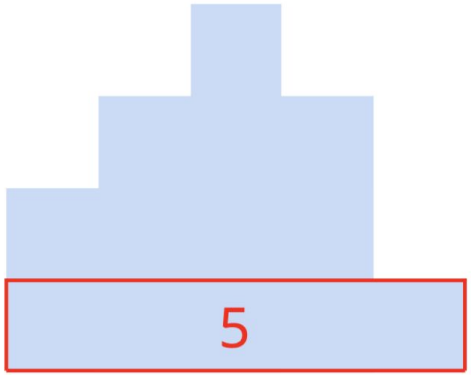
例題三、長條圖最大矩形

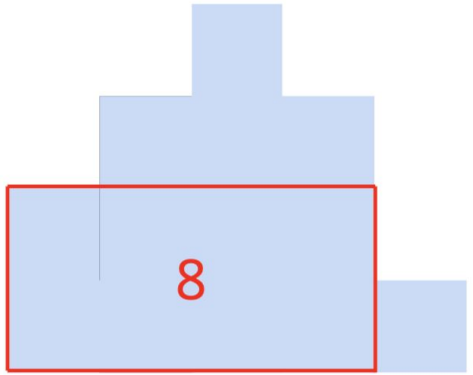
題目敘述

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 <= 10^5、高度 <= 10^9）

範例：

2 3 4 3 1



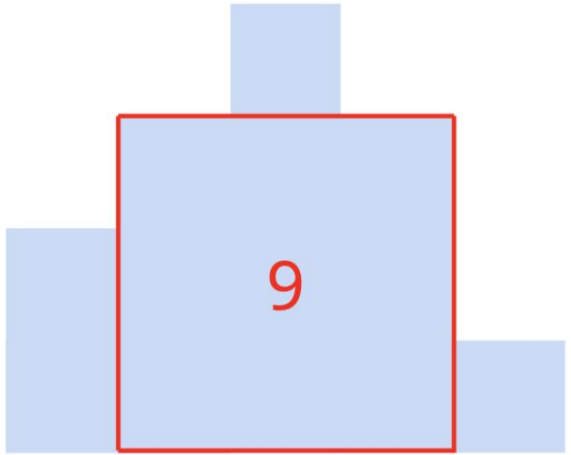


題目敘述

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 <= 10^5、高度 <= 10^9）

範例：

2 3 4 3 1



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！

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

```
while (size() > 0 && top() >= value[idx]) { // top比我還大
    pop(); // 把top丟掉
}
if (size() > 0) ans[idx] = top();
// 目前的top會是比我小且離我最近的那個
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」

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 <= 10^6）

Linked-list

Linked-list 的概念

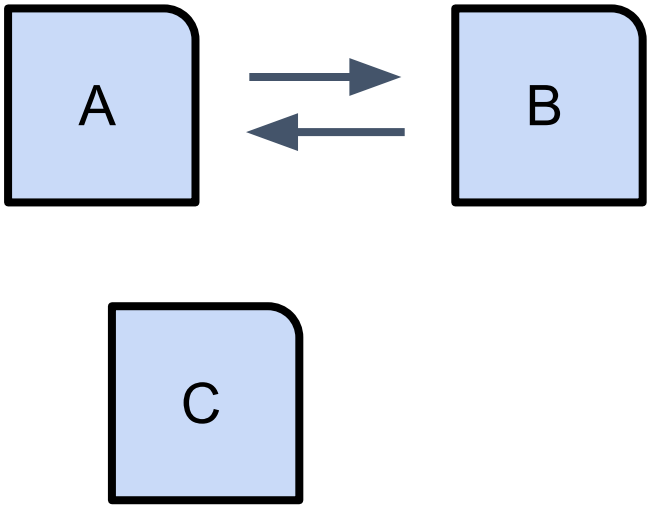
- 對於每個資料紀錄前後資料的位置
- 可以 O(1) 加入、刪除特定資料
- 不支援 random-access
  - 不能 O(1) 存取指定 index 的資料

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？

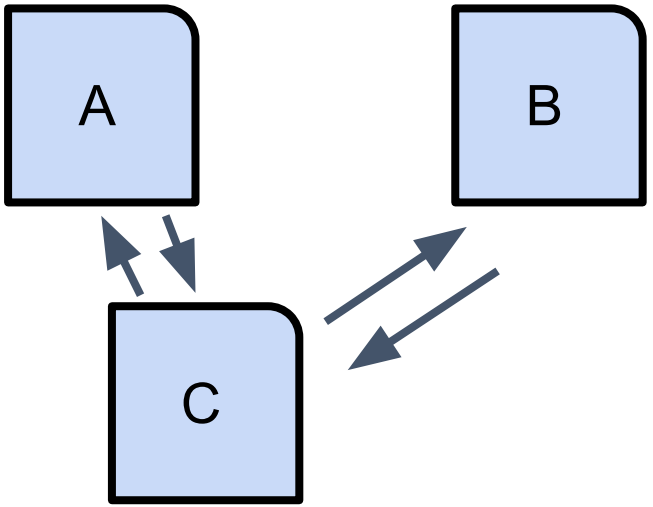
加入資料

- 假設我們想將資料 C 插入在資料 A、B 之間



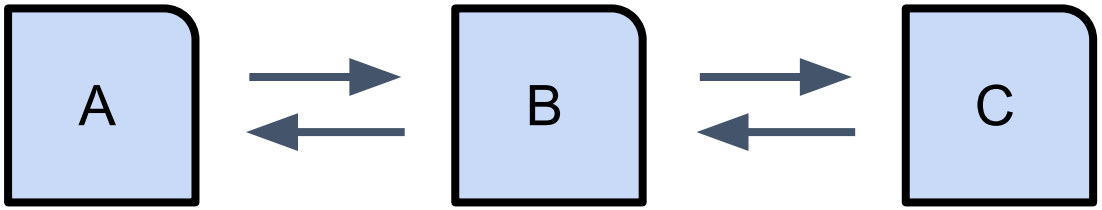
加入資料

- 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



實作

- 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();
}
```

另外一種作法

- 使用陣列的 index，又稱偽指標

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

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

Try it

- Try NEOJ #20, #24
- #20 Hint: https://pastebin.com/sWRRxrpz

謝謝大家！

Sprout