STACKS & QUEUES

Prof. Michael Tsai

2013/3/5

http://ppt.cc/Cjly

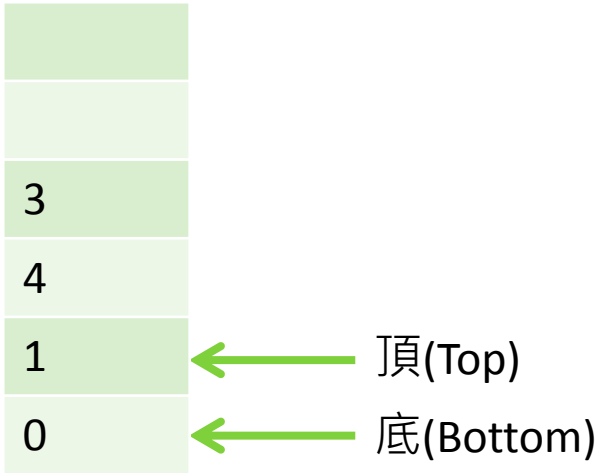
Stack

- Stack是蝦密碗糕?

- 一串有”順序”的列表
- 又叫Ordered List

- 有什麼特別的地方?

- 永遠從頂部拿與放
- 拿: Pop
- 放: Push
- 看一個例子
- 所以先放進去的, 會??出來?



A:後

Stack

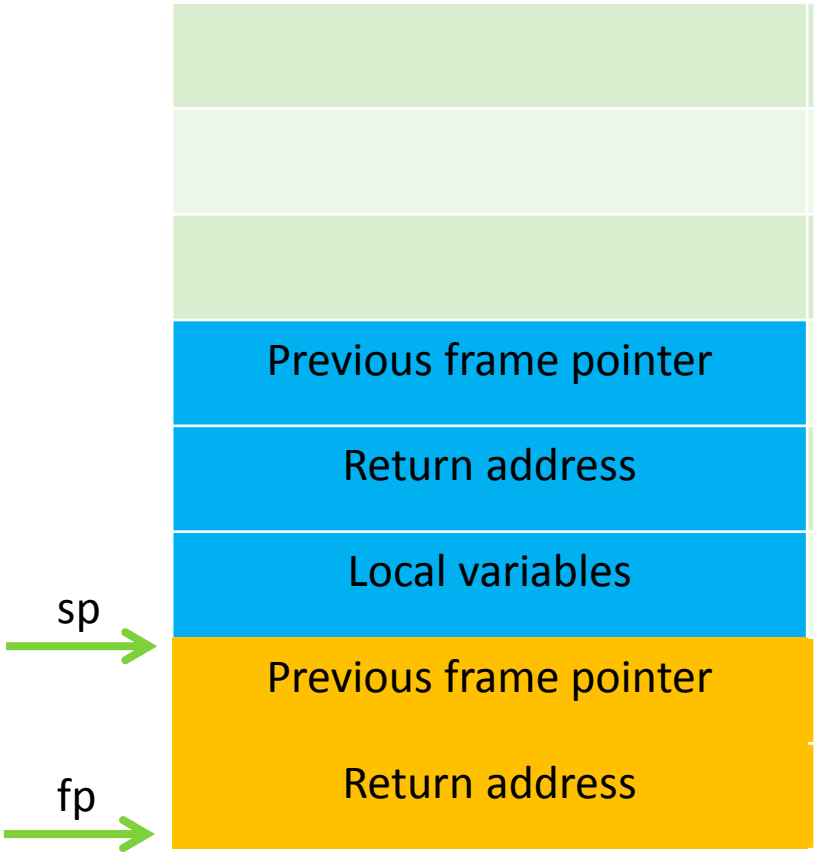
- 所以是一個先進後出(First-In-Last-Out, FILO)的資料結構
- 或是後進先出(Last-In-First-Out, LIFO)

- 那麼, 支援那些動作呢? 各動作又需要什麼參數?
- 請同學們列舉:

- 做一個Stack
- 放(Push)
- 拿(Pop)
- 是不是空的?
- 是不是滿了?
- 最上面的元素(不pop出來)
- 總共有幾個元素

例子一: 系統堆疊

- 被程式拿來儲存呼叫function的相關資訊
- 放什麼? 叫做activation record或stack frame
  - return address: 呼叫function的下一個指令應該去的地方
  - previous frame pointer: 指到前一個stack frame在stack裡面的位置(底)
  - local variables
- 看一個例子.
- 適不適用recursive call?
- Stack overflow?



怎麼implement一個stack?

- 請同學上來舉例說明☺
- 先試試看使用大家都會的array
- 除了array以外, 還需要哪些東西?
  - 紀錄頂部的index
- 怎麼implement這些operations?
  - Push
  - Pop
  - 是不是空?
  - 是不是滿?
  - 最上面的元素(不pop出來)
  - 總共有幾個元素

Code

```
struct ArrayStack {
	int top;
	int capacity;
	int *array;
};

struct ArrayStack *CreateStack() {
	struct ArrayStack *S = malloc(sizeof(struct
ArrayStack));
	if (!S) return NULL;
	S->capacity=4;
	S->top=-1;
	S->array=(int*)malloc(S->capacity*sizeof(int))
	if (!S->array) return NULL;
	return S;
}
```

Code

```
void Push(struct ArrayStack*S, int data) {
      if (IsFullStack(S))
            printf("Stack OverFlow");
      else
            S->array[++S->top]=data;
}

int Pop(struct ArrayStack*S) {
      if (IsEmptyStack(S)) {
            printf("Stack is Empty");
            return 0;
      } else
            return (S->array[S->top--]);

}
```

滿了怎麼辦?

- `array=(int*)malloc(S->capacity*sizeof(int));`

- 如果裡面有超過capacity個元素, 就滿了. T_T
- 如何把stack變大?

- 複習: realloc做什麼用的?

- realloc做了什麼? (假設原本大小n, 新的大小m)
  1. 在記憶體某個地方開一塊新的記憶體, 大小為m(比較大)
  2. 把放在舊的記憶體的東西搬到新的記憶體去 (需花n的時間)
  3. 然後把新的記憶體位址回傳給你

- realloc相關決定:
- 要realloc成多大?
- realloc完畢有沒有什麼要改的地方?

來一點複雜度分析

小名詞: Amortized Analysis
把n個動作的執行時間一起考慮, 看平均起來它們總共要花多少時間執行及一個動作平均要花多少時間, 這樣的分析方法叫做Amortized Analysis (下學期algorithm會教到)

- 方法一:
- 當滿了以後, 每次push就把array大小增加1,
- 每次pop就把array大小減少1
- 問題: push n 次的時間複雜度為多少? (假設一開始是空的)
- 假設:
- Push**一個**element是O(1),Allocate**一個**element是O(1),移動**一個**element是O(1)

- 第一次Push (花constant時間)
- 第二次Push (花constant時間+花**1**的時間把舊記憶體的東西搬到新記憶體)
- 第三次Push (花constant時間+花**2**的時間把舊記憶體的東西搬到新記憶體)
- 第四次Push (花constant時間+花**3**的時間把舊記憶體的東西搬到新記憶體)
- …
- 第n次Push (花constant時間+花**n-1**的時間把舊記憶體的東西搬到新記憶體)
- 1+2+…+(n-1)=$\frac{(n-1)(1+(n-1))}{2}$=$O(n^2)$

來一點複雜度分析

- 方法二:
- 如果每次都realloc成原本的兩倍大
- capacity 存現在的stack大小
- 問題: push n 次的時間複雜度為多少?

- 第一次Push (花constant時間)
- 第二次Push (花constant時間+花**1**的時間把舊記憶體的東西搬到新記憶體)
- 第三次Push (花constant時間+花**2**的時間把舊記憶體的東西搬到新記憶體)
- 第四次Push (花constant時間)
- 第五次Push (花constant時間+花**4**的時間把舊記憶體的東西搬到新記憶體)
- …
- 第九次Push (花constant時間+花**8**的時間把舊記憶體的東西搬到新記憶體)

- 做了n次push以後, 花多少時間?

來一點複雜度分析

- Constant time的部分共n (因為有n個O(1))
- 剩下把舊的記憶體裡面的資料搬到新的記憶體裡面去的時間
- $1 + 2 + 4 + \cdots + 2^{\lfloor \log_2 n \rfloor}$

$$\leq 1 + 2 + 4 + \cdots + 2^{\log_2 n}$$

$$= \frac{1\left(2^{\log_2 n} - 1\right)}{2 - 1} = O(n)$$

- 所以總共n個push花O(n)的時間!
- 平均起來一個push花O(1)的時間!

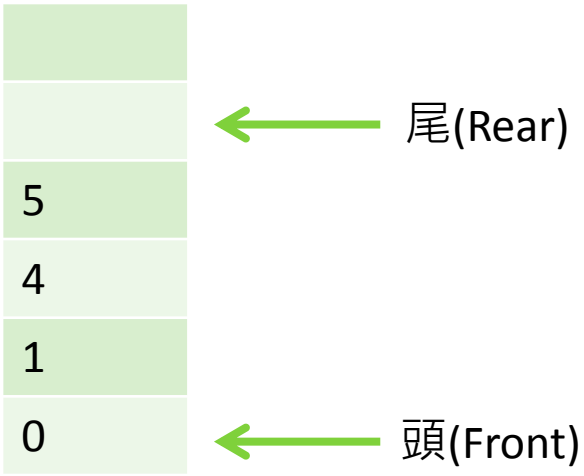
- 詳細的程式碼請看Karumanchi Ch4 P.77 & P.79
- <動腦時間: 回家作業> 如果每次增加為原本capacity的c倍, c>1, 那麼複雜度會有變化嗎? 為什麼?

答案:不會

Monitor Stack

Queue

- Queue是蝦密咚咚?
- 也是Ordered List
- 和Stack有什麼不一樣?
- 從尾(Rear)放, 從頭(Front)拿
- 拿: Delete, or DeQueue
- 放: Add, or EnQueue
- 所以先放進去的, 會??出來?



Queue

- 先放進去的, 會先出來
- 所以是First-In-First-Out (FIFO)

- 那麼, 支援那些動作呢? 各動作又需要什麼參數?
- 請同學們列舉:

- 做一個Queue
- 放(EnQueue)
- 拿(DeQueue)
- 是不是空的?
- 是不是滿了?
- (其他的: 最前面的元素是什麼, 總共有幾個元素)

舉一個例子

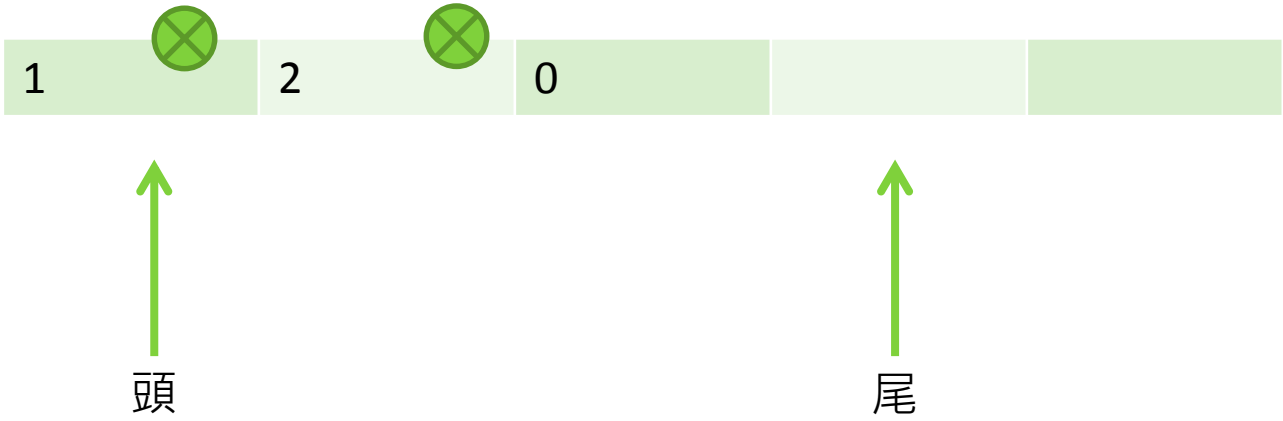
- Job scheduling
- 如果沒有使用priority的概念的話
- 先來的job就先做 → FIFO的概念

- 問題:程式要求作業系統做一個job, 作業系統要照來的順序決定誰先使用資源.
- 作法:
- 程式發出要求的時候, 就放入queue中
- 資源空出來了, 就從queue拿出一個job來做
- 結果: FIFO
- <動腦時間> 如果放入一個stack會怎麼樣?

那麼, 怎麼implement queue呢?

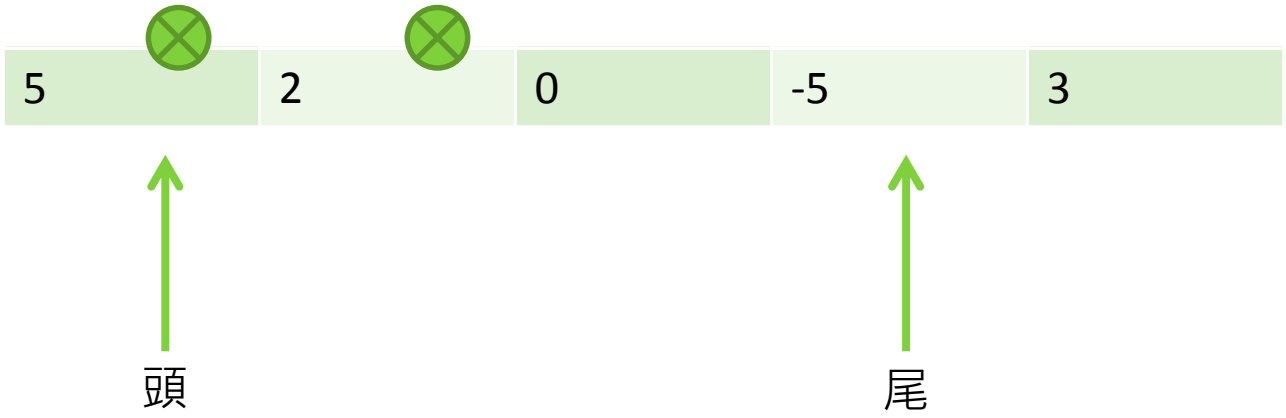
- 一樣使用array
- 要記得頭跟尾的在array裡面的位置(index)

- 但是這次有點問題



- 滿了! 但是前面的空間沒有用到!
- 怎麼解決?

解決方式: 頭尾相接



- 稱為Circular Array Implementation
- 這樣就可以保證queue儲存的element數目最多可以跟array大小一樣

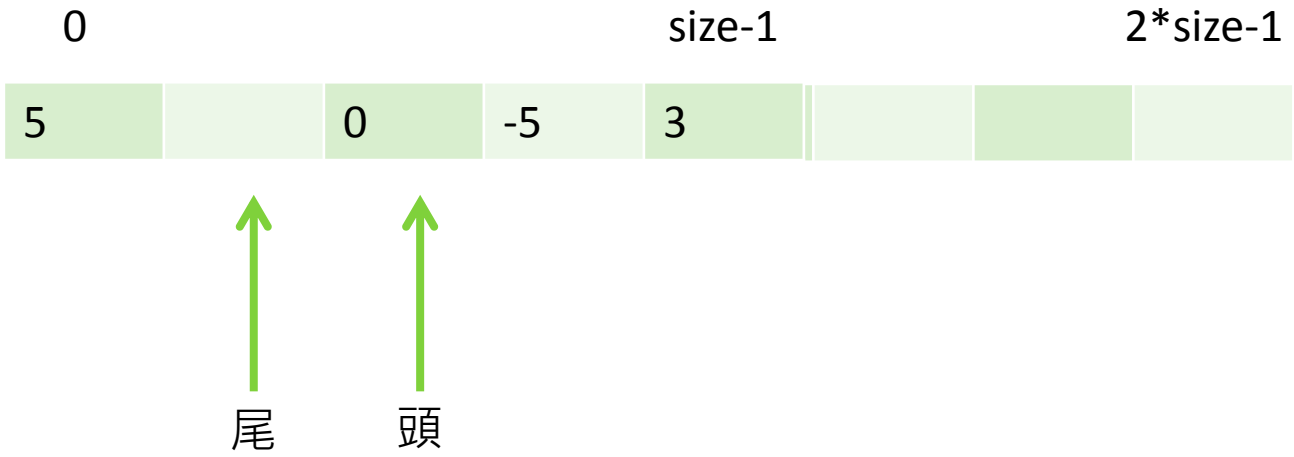
- 什麼時候是空?
- 什麼時候是滿?
- (能不能分辨空和滿?)

那麼, 如果要使用動態大小呢?

- 一樣使用realloc來要到一塊更大的記憶體

- 然後還有什麼工作要做?
- realloc之後的樣子: 看起來不太對

realloc from **size** to **2*size**



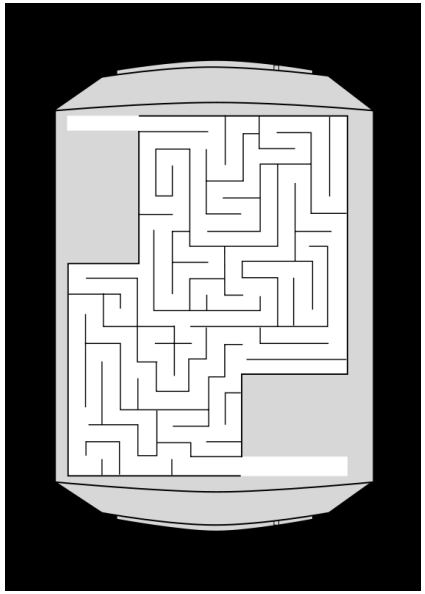
- (答案請看Karumanchi課本P.101 的ReSizeQueue()怎麼寫的)

接下來講一些應用:

- 計算機問題

- 迷宮問題

(回家自己看)



應用一: 計算機

- 不是普通的計算機
- 題目: 如果打入一串如1+2*3-5/(4+5)/5的算式, 請寫一個演算法來算出這串算式的結果.
- 怎麼寫呢? 好複雜T_T

先來看看算式長什麼樣子

- 1+2*3-5/(4+5)/5

- 裡面有:
- Operand – 1, 2, 3, 5, 4, 5, etc.
- Operator – + * - /
- 括號– (,)

- 特色1: 左到右(left-to-right associativity)
- 特色2:一般這種寫法叫做infix (operator夾在operand中間)
- 先後順序要operator去”比大小”
- 例如乘除是大, 加減是小, 那麼就先乘除後加減

別種寫法: postfix

- 把operator放到兩個operands的後面
- 例如 2+3*4 →2 3 4 * +
- a*b+5 → ?
- (1+2)*7 → ?
- a*b/c →?
- (a/(b-c+d))*(e-a)*c→?
- a/b-c+d*→?
- e-a*c→?

Postfix有什麼好處?

- 沒有括號
- 用stack幫忙就可以很容易地算出結果!

- 例子: 6 2 / 3 – 4 2 * +
- 從左邊讀過去
- 讀到6, 放6進stack (stack: 6)
- 讀到2, 放2進stack (stack: 6 2)
- 讀到/, 取兩個operands (6和2), 算6/2, 然後答案放回去stack (stack: 3)
- 讀到3, 放3進stack (stack: 3 3)
- 讀到-, 取兩個operands(3和3), 算3-3, 然後答案放回去stack (stack: 0)
- 依此類推....(請同學上來繼續完成☺)

剩下來的”小問題”: infix to postfix

- “小問題”: 怎麼把infix expression轉成postfix expression?

- 第一種方法: (適合紙上談兵)
- 1. 把整個expression
- 2. 把所有的operator都移到operand的後面去, 方便去除所有括號
- 3. 去除所有括號

- 但是實作上要怎麼做呢? (不希望做兩次)

又是一個可以利用stack解決的問題

- 方法:
- 1. 從左至右讀取expression
- 2. 碰到operand就直接輸出
- 3. 碰到operator時比較stack頂上的operator和目前讀到的operator哪一個比較”大” (precedence)
  - 如果目前讀到的operator**比較大**, 就把operator放到stack裡
  - 如果**一樣大**或者現在讀到的operator**比較小**, 就一直把stack裡面的operator拿出來印出來, 一直到stack是空的或者現在的operator比stack頂的operator大

- 為什麼可以這樣做?
- 裡面括號需要先印出, 但是卻是後讀到
- 比較外面的先用stack記起來, FILO, 由內而外

例子一

- 請一位同學來解說 怎麼把a+b*c利用前述方法轉換成postfix ☺
- 那麼, 如果碰到括號怎麼辦?

括號

- 括號內的有優先性

- 因此當碰到右括號的時候, 就立刻把stack中的東西一直拿出來直到碰到左括號為止
- 舉例: a*(b+c)
- 輸出的內容, stack內容(最右邊為top)
- a
- a, *
- a, *(
- ab, *(
- ab, *(+
- abc, *(+
- abc+*

還有一些小問題

- 小問題1: 左括號的”大小”到底是多少?
- 碰到左括號一定要放進去stack →此時要是最大的
- 左括號的下一個operator一定要可以放進去→此時要是最小的

- 結論:
- 左括號有兩種”大小”
- 在stack內的時候優先性為最小
- 在stack內的時候優先性為最大
- 其他operator的優先性則不管在stack內外都一樣

還有一些小問題

- Q: 如何確保stack空的時候, 第一次碰到的operator可放進去?
- A: 在stack底部放入一個虛擬operator帶有最低的優先性
- Q: 如何確保讀到expression最後的時候, 可以把stack裡面未取出的operator都拿出來?
- A: 字串最後可以加一個優先性最低的虛擬operator

例題

- a+(b*c/(d-f)+e)
- 用白板解說 (請同學? :P)

答案: abc*df-/e++

最後來一個有趣的迷宮問題吧

- 迷宮: 0是路, 1是牆壁. 每一部可以往上、下、左、右和四個斜角方向走一步.
- 問題: 怎麼找出一條路從(0,0)走到(7,7)？(不一定要最短)
- 提示: 跟stack是朋友

	0	1	2	3	4	5	6	7
0	0	1	1	1	1	0	1	1
1	0	0	0	0	0	0	0	1
2	1	1	1	1	1	1	1	0
3	1	0	0	0	0	0	0	1
4	0	1	1	1	1	1	1	1
5	0	0	0	1	1	0	1	0
6	0	1	0	1	0	1	0	1
7	0	1	1	0	1	1	1	0

走迷宮的時候, 人要怎麼走?

- 最重要的時候, 是碰到岔路的時候
- 先記起來, 選其中一條走走看
- 如果碰壁了 (一直沒有走到終點), 就退回最後一次碰到的岔路, 換另外一條岔路
- 關鍵字: 最後一次碰到的岔路 (不是最先碰到的)
- 所以是先進後出→使用stack
- 關鍵字: 換另外一條岔路
- 要記得上一次走過哪一條路了

一些細節

- Q: 那麼, stack裡面要存什麼呢?

- A:
- “岔路”的地方的
- 座標, 也就是(row, col)
- 試過那些岔路了(試到八個方向的哪個方向了)

- Q: 要怎麼預防繞圈圈? (永遠出不來)
- A: 標示所有已經走過的地方, 走過就不用再走了
- <注意> 這是因為不用找”最短”的路

讓我們來寫algorithm

- 把(row_start, col_start, 第一個方向) 放入stack
- while(stack不是空的) {
  - 從stack拿出一組岔路點(row, col, dir)
    - while(還有別的dir還沒試) {
      - 將(row, col)往dir方向移動, 得到(row_n, col_n, dir).
      - 如果(row_n, col_n)就是終點, 則結束
      - 如果(row_n, col_n)不是牆壁且沒有來過 {
        - 標示(row_n, col_n)來過了
        - 把(row, col, dir的下一個方向)放入stack
        - row=row_n; col=col_n; dir=第一個方向;
      - }
    - }
- }

Today's Reading Assignments

1. Karumanchi 4.1-4.6 5.1-5.6
   (如果有空的話自己閱讀4.7 or 5.7上的問題, 看看自己觀念是否都懂了!)
2. 本份投影片上沒有講的部分