1 倍增算法

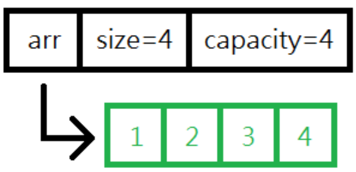
有時候我們常常可以透過「把問題切一半遞迴處理」來獲得問題的解法或者更快速的算法，如之後會提到的分治算法、merge sort 等等。然而，反過來說，如果我們把已知的小範圍方法透過「把解法放大一倍遞推處理」，有時也會有意想不到的效果。以下我們用動態陣列 (dynamic array) 來介紹倍增算法的應用。

在許多情況下，我們在儲存資料之前沒辦法知道資料的總量，從而很難決定陣列宣告的大小。如果一開始宣告的陣列大小不夠，就很容易存取到超出陣列範圍的記憶體而發生不預期的結果；反之如果一開始就宣告很大的陣列範圍，則會造成記憶體的浪費，甚至超過限制的記憶體大小。如果只需要存取陣列頭尾的元素的話，可以使用之前教過的 linked list 實做，但如果需要支援隨機存取 (random access)，就不能使用 linked list 了。此時動態陣列就派上用場，可以想成它是一個「大小會自己伸縮的陣列」，以下為一種只考慮新增元素至陣列尾端，不考慮刪除元素的動態陣列實做方法 (意即，只會變長，不會縮短)：

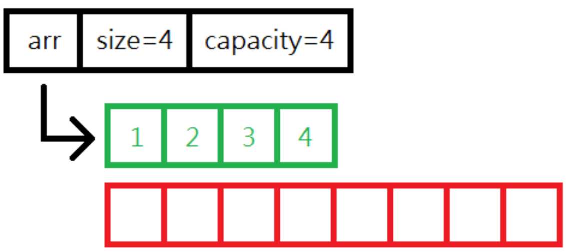
- 記錄一個陣列的指標，當前宣告的陣列的大小 (`capacity`)，以及當前在陣列中的元素個數 (`size`)。該陣列指標指向的陣列是真正存放元素的地方。初始化時，$\texttt{capacity} = 1, \texttt{size} = 0$。
- 新增一個元素時，若當前的元素個數未達上限 ($\texttt{size} < \texttt{capacity}$)，則直接將該元素放進陣列尾端，並將 `size` 增加 1。若元素個數已達上限，則動態宣告 (C 中的 malloc 或 C++ 中的 new) 一個大小為 $\texttt{capacity} \times 2$ 的陣列，並用 $O(\texttt{capacity})$ 的時間將當前陣列的所有元素都複製過去新陣列，接著釋放原陣列的記憶體 (C 中的 free 或 C++ 中的 delete)，最後再放入新增的元素，將 `size` 增加 1。

以下以圖示說明新增元素的操作：

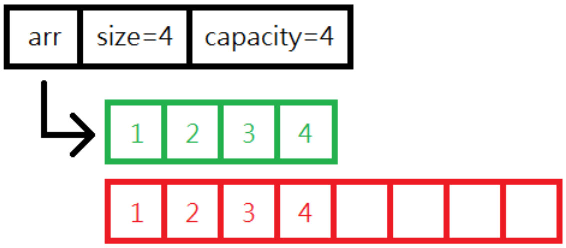
1. 假設當前的容器內有 `size`=4 個元素：(1,2,3,4)，且 `capacity`=4



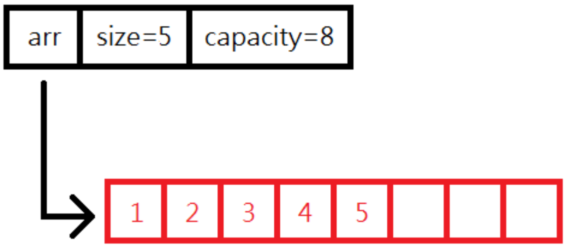
2. 當要新增元素「5」時，發現已經沒有空間了，因此先宣告一個大小為原本兩倍的陣列



3. 將原本陣列中的元素一個一個搬到新的陣列中



4. 最後將原本的陣列刪除，指標指向新陣列，並將「5」新增到新陣列中



動態陣列的操作複雜度將在作業中請大家證明，以下連結是一份只能從尾端插入元素的動態陣列程式碼。

Link: dynamic_array.c , dynamic_array.cpp

順帶一提，C++ STL 中的 vector 容器即是動態陣列，是個好用的資料結構。以下簡單介紹一些 std::vector 的常用函數：

- size()：容器內的元素個數
- empty()：容器是否是空的
- clear()：清除所有容器內的元素
- push_back(x)：新增元素 x 至容器末端
- pop_back()：刪除容器末端的元素

習題

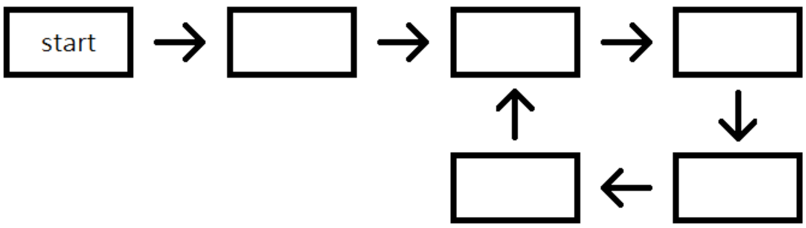
1. 了解動態陣列的原理之後，請推導新增元素時的複雜度。(malloc/new 的時間可以不考慮)

    (a) (20 pts) 請證明一個動態陣列若從初始狀態開始進行了 n 次的新增元素操作，總時間複雜度為 $O(n)$，空間複雜度也是 $O(n)$。

    (b) (10 pts) 請證明一個動態陣列若從初始狀態開始進行了 n 次的新增元素操作，但擴張陣列時，大小不是增加到 `capacity` $\times 2$，而是 `capacity` $+ 1$，則總時間複雜度為 $O(n^2)$，空間複雜度是 $O(n)$。

2. 有 n 個節點，每個節點內只存著它的下一個節點。已知從 start 開始，每次走到當前節點的下一個節點，最終可以走過所有的節點，並且進入一個環。也就是說，這些節點形成的形狀就像字母 ρ 一樣。

```c
struct Node {
    struct Node* next;
};
struct Node* start;
```



請在不知道 n 的確切數值之下，找出這個環的長度。(意即，不能有如「從 start 開始走 n 步」的敘述，因為你並不知道 n 是多少)

    (a) (10 pts) 請給出時間複雜度 $O(n)$，不限制空間大小的做法。(額外空間宣告在 Node 裡面或外面皆可)

    (b) (20 pts) 給定 k，且假設環和起點的距離小於 k，請描述如何判斷環的大小是否小於 k，如果小於 k 的話還要給出環的大小。另外，限制時間複雜度為 $O(k)$，(額外空間的) 空間複雜度為 $O(1)$。(環和起點的距離為 d 表示從起點開始至少走 d 步之後會進入環)

    (c) (20 pts) 令 k 從 1 開始，檢查 (b) 中提到的判斷是否成立，如果成立則可以得到環的長度，否則將 k 變成 2 倍，繼續判斷直到得到環的長度。請證明這個演算法一定會停止，並證明該演算法的時間複雜度為 $O(n)$，(額外空間的) 空間複雜度為 $O(1)$。

Note: (b) 跟 (c) 中不允許修改 Node 中的資料。
Hint: 縱使不能修改 Node 本身，但你可以記錄指標的值來記下某一個走過的點。

3. 以下為 2015 年資訊之芽入芽考的其中一題。

Description

由於円円嚴重缺乏運動，他的好朋友們幫他設計了一個好玩的跳格子遊戲，讓他在娛樂之餘還能順便活動身體，希望能讓他再長高一點 (雖然應該希望渺茫了)。

這個跳格子的遊戲是這樣的：一開始在地上畫出一條 $1 \times N$ 的方格圖，並且大小依序標上編號 0 ~ $(N-1)$。接著在每個格子裡面寫上一個數字 a_i，表示當円円跳到第 i 格之後，下一次就要跳到第 a_i 格。由於在格子內轉身不太方便，因此円円的好朋友們十分好心，填上數字時一定保證 $a_i \geq i$，也就是說，円円只會往前跳或是待在原地，而不會往後跳。

現在已知円円一開始在第 x 格，請問他跳了 k 步之後會停在哪格呢？

Input

第一行為兩個正整數 N, Q，表示共有 N 個格子，且有 Q 次詢問。
第二行為 N 個整數，依序為 $a_0, a_1, \cdots, a_{N-1}$。
接著 Q 行，每行為兩個正整數 x, k，表示円円一開始在第 x 個格子，並且接著要跳 k 步。

- $0 \leq x, k \leq (N-1)$

(a) (5 pts) 如果對於每筆詢問 (x, k)，都一步一步的跳 (所謂的一步一腳印) 得到最後的答案，時間複雜度為何？(請以 N, Q 表示)

(b) (10 pts) 若以 $a[i]$ 表示從第 i 格跳 1 步之後所到達的格子編號 (即題目中的 a_i)，請以 (a, i) 表示從第 i 格跳 2 步之後所到達的格子編號。

(c) (10 pts) 承上題，若以 $b[i][j]$ 表示從第 i 格跳 2^j 步之後所到達的格子編號，請以 $(b, i, j-1)$ 表示 $b[i][j]$ 在 $j > 0$ 時的遞迴關係。
（解答的長度為 $O(1)$，例如：不能寫重複哪個式子 2^j 遍）

$$b[i][j] = \begin{cases} a[i] & , \text{ if } j = 0 \\ ??? & , \text{ if } j > 0 \end{cases}$$

(d) (10 pts) 承上題，假設已經建好了 $b[i][j]$ 陣列，給定 x, k，請在 $O(\log k)$(或是 $O(\log N)$) 的時間內得到從 x 開始跳 k 步之後所在的格子編號。

(e) (5 pts) 承上題，請問上述演算法的時間複雜度為何？(請以 N, Q 表示)