1 倍增算法

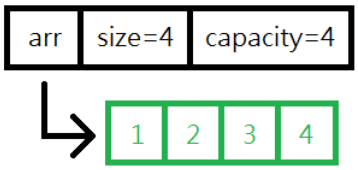
有時候我們常常可以透過「把問題切一半遞迴處理」來獲得問題的解法或者更快速的算法，如上課中提到的分治算法、merge sort 等等。然而，反過來說，如果我們把已知的小範圍方法透過「把解法放大一倍遞推處理」，有時也會有意想不到的效果。以下我們用動態陣列（dynamic array）來介紹倍增算法的應用。

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 實作，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）：

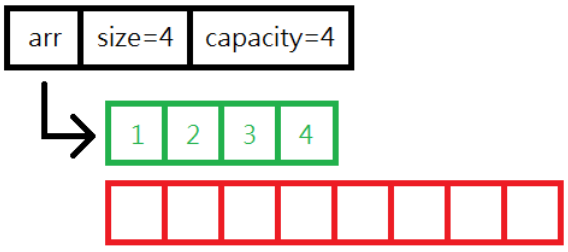
- 記錄一個陣列的指標，當前宣告的陣列的大小（`capacity`），以及當前在陣列中的元素個數（`size`）。該陣列指標指向的陣列是真正存放元素的地方。初始化時，$\texttt{capacity} = 1, \texttt{size} = 0$。
- 新增一個元素時，若當前的元素個數未達上限（$\texttt{size} < \texttt{capacity}$），則直接將該元素放進陣列尾端，並將 `size` 增加 1。若元素個數已達上限，則動態宣告（C 中的 `malloc` 或 C++ 中的 `new`）一個大小為 $\texttt{capacity} \times 2$ 的陣列，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：

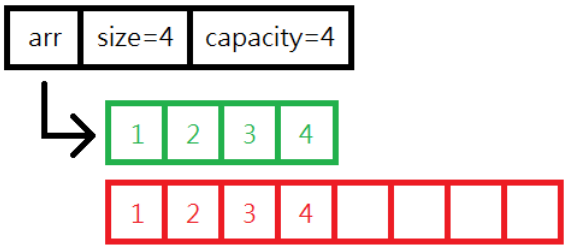
1. 假設當前的容器內有 `size`=4 個元素：(1,2,3,4)，且 `capacity`=4



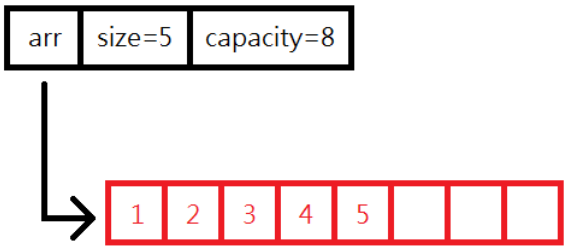
2. 當要新增元素「5」時，發現已經沒有空間了，因此先宣告一個大小為原本兩倍的陣列



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. 了解過動態陣列之後，請推導新增元素時的複雜度。（不考慮 `malloc/new` 的時間）

   (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. 以下為 2015 年資訊之芽入芽考的其中一題。注意到在回答複雜度時，你總是要考慮最糟糕的情況（Worst case）。

Description

由於円円嚴重缺乏運動，他的好朋友們幫他設計了好玩的跳格子遊戲，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（雖然希望渺茫）。

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 $0 \sim (N-1)$。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 d 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, d，表示円円一開始在第 x 個格子，並且接著要跳 d 步。

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

   (a) (5 pts) 如果我們預處理存下每個人往前跳一步、兩步、三步、……、N 步的答案，那詢問就可以很快樂的 $O(1)$ 回答，可惜預處理時間卻是慘不忍睹的 $O(N^2)$；反之，如果我們每次詢問都一步一腳印的一步一步跳，預處理時間就是 $O(1)$，但時間就相對的退化到了 $O(N)$。

   我們不妨在其中找到一個平衡點：如果我們只預處理存下至多 K 步的答案呢？如此一來，如果一筆詢問為 (x, d)，我們就能夠先找到 x 跳 K 步的格子 y，再找到 y 跳 K 步的格子 z，……，直到剩餘的步數不超過 K，就可以找到答案。好像稍稍變快了，請回答這個演算法的預處理時間複雜度和總詢問時間複雜度。（請以 N, Q, K 表示，注意詢問次數是 Q！）

(b) (5 pts) 假設 $O(Q) = O(N)$，我們應該要設 K 為多少才能讓預處理時間複雜度和總詢問時間複雜度相等呢？（請以 N 表示，並說明你得到答案的過程）

(c) (5 pts) 雖然前一個小題的複雜度好像還不夠好，可是已經有很大的改進了！既然詢問時我們會 K 個 K 個跳，不妨我們再貪心一點，再次預處理存下每個格子往前跳 K 步、往前跳 $2K$、往前跳 $3K$ 步、……、往前跳 K^2 步的答案，也就是說，預先存下跳至多 K 個 K 步的答案。

不過這個表要怎麼建啊？假設 $jump[i][j][0]$ 是第 i 個格子往前跳 j 步的答案；$jump[i][j][1]$ 是第 i 個格子往前跳 j 個 K 步的答案，$jump[i][j][0]$ 我們會建，$jump[i][j][1]$ 的話得考慮下列這個遞迴式：

$$jump[i][j][1] = \begin{cases} jump[i][K][0] & , \text{ if } j = 1 \\ ??? & , \text{ if } j > 1 \end{cases}$$

請給出上述的 ???，使得他是一個 $O(1)$ 的式子。（意及你不能寫重複哪個式子 K 遍這種敘述）

(d) (10 pts) 咦？這樣不就能每次跳 K^2 步，然後在距離小於 K^2 的時候再跳某幾個 K 步，最後就能讓距離小於 K 用一開始的表格找到定位嗎？預處理的部份只要用 j 遞增的順序去算，就可以每次 $O(1)$ 得到每個欄位的數值了吧！

假設 $O(Q) = O(N)$，請以 N, K 表示上述演算法的預處理時間複雜度和總詢問時間複雜度，並給出一個 K 使得預處理時間複雜度和總詢問時間複雜度相等。

(e) (10 pts) 原來分兩層這麼有用……不然這樣好了，分 p 層如何？也就是說：

在第三層存下每個格子往前跳 K^2 步、$2K^2$ 步、$3K^2$ 步、……、K^3 步的答案。

在第四層存下每個格子往前跳 K^3 步、$2K^3$ 步、$3K^3$ 步、……、K^4 步的答案。

……

令 $jump[i][j][l]$ 代表第 l 層裡，從 i 往前跳 j 個 K^l 步的答案，那就有遞迴式：

$$jump[i][j][l] = \begin{cases} ???_1 & , \text{ if } j = 1 \\ ???_2 & , \text{ if } j > 0 \end{cases}$$

只考慮 $l > 0$，請給出上述的 $???_1, ???_2$，使得他們都是一個 $O(1)$ 的式子。

(f) (15 pts) 承上題，假設 $O(Q) = O(N)$，請給出預處理時間複雜度後，簡單說明利用給定表格的詢問方法、以及總詢問時間複雜度。（請用 N, K, p 表示，注意你不可以把 p 當常數！）

(g) (5 pts) 承上題，如果 p 取得足夠大的話，其實 K^p 就大於 N 了，也就沒有一口氣跳 K^p 步的必要；相對的，可以發現 K 最小最小也只能是 2，否則就失去了預處理的意義。

不過 $K = 2$ 的時候好像好處多多？畢竟在前述的問題當中，我們發現有些欄位的值重複了，非常冗餘。如果 $K = 2$，則 j 只有兩種值，那不就可以藉著這些重複少存開一個維度嗎？

請給出一個盡量小的 p 使得 $2^p > N$。

(h) (15 pts) 承上題，正確來說，遵循著這個架構所產生出來的表格是 $b[i][j]$，代表第 i 個格子往前跳 K^j 也就是 2^j 的答案！而 $b[i][j]$ 有著下列遞迴式：

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

給出上述的 ??? 使它是一個 $O(1)$ 的式子，並繼承前一小題你所回答的 p 值，在假設 $O(Q) = O(N)$ 的情況下，給出預處理時間複雜度，並說明基於該表格詢問的演算法和總詢問時間複雜度。

你是否發現了倍增法的產生過程呢？你的兩個時間複雜度是不是很自然的一樣了呢？（這行不是問題，請不要回答）