多元學習表現

APCS 大學程式設計先修檢測

姓名：吳子龍

日期：2022 年 10 月 23 日

目錄

1 參加動機 **4**

2 檢定資料 **4**

2.1 題目題型 4

2.2 題目範圍 4

2.3 設備 5

3 檢定訓練 **5**

3.1 第一月 5

3.1.1 DFS 5

3.1.2 BFS 5

3.1.3 stack 6

3.1.4 queue 6

3.1.5 set 6

3.1.6 map 6

3.2 第二、三月 7

3.2.1 搜尋 7

3.2.2 枚舉 7

3.2.3 貪心 8

3.2.4 分治 8

3.2.5 數論 8

3.2.6 動態規劃 9

3.2.7 圖 9

3.2.8 前綴和與差分 10

3.2.9 BIT 樹 10

3.2.10 線段樹 11

3.2.11 並查集 11

3.3 第四、五月 11

3.3.1 遞迴 11

3.3.2 滑動窗口 11

3.3.3 掃描線 11

3.3.4 set、map 11

3.3.5 bitset 12

3.3.6 stack 12

3.3.7 deque 12

4 結論 **12**

4.1 分數 12

4.2 檢討 13

4.3 心得 13

1 參加動機

程式一直以來一直是我最興趣也最想要鑽研的科目，雖然在電機科沒有程式的課程，但我還是在課外的時候找出時間來自我練習，但我總覺的我必需要有一個檢定來測試自我，所以我就在我的老師的推薦下參加了APCS。

2 檢定資料

2.1 題目題型

題型分為實作題與觀念題，在一天內應試完畢，實作題考試時間為 2 小時又 30 分鐘，觀念題考試時間為 2 小時。實作題為後測評分，在測試時只有範例測資可以進行測試。

實作題分為 4 題，題目難度也是由簡單到難，越靠前的題目越不需要使用複雜的演算法或資料結構，但在實作上的複雜度也會更加的麻煩。

觀念題分為兩節考試，題目難度並沒排序，因此要在一定的時間內解出越多題越有利，也需要練習程式的追蹤，以及在紙上練習計算。

2.2 題目範圍

題目以大學程式為基礎設計，包含了兩個部份，演算法及資料結構。基本題型包含：

- 輸入與輸出
- 算術運算、邏輯運算、位元運算
- 條件判斷、迴圈
- 陣列與結構
- 字元、字串
- 函數、遞迴

演算法包含：

- 枚舉
- 排序
- 搜尋
- 分治
- 貪心
- 動態規劃

- DFS、BFS

資料結構包含：

- 佇列
- 堆疊
- 樹狀圖
- 圖

2.3 設備

檢定的程式語言可選用 C、C++、Java、Python，以下以 C++ 為例。

- 作業系統：Lubuntu Desktop 18.04 (64-bit)
- 編輯器：Code::Blocks 17.12
- 編譯器：Gcc 7.3.0
- C++ 版本：C++11

3 檢定訓練

因為在學校沒有相關的課程，所以我使用網路上的資源進行自學，大約訓練了五個月。

3.1 第一月

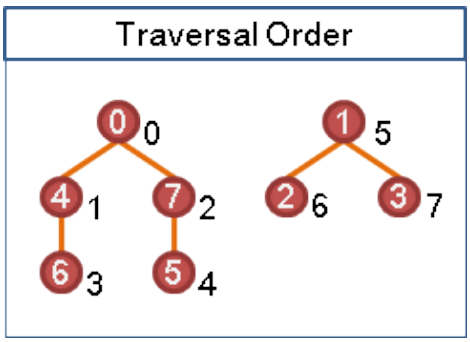
前一個月主要為複習我以前所學的知識，以及大至上的了解考試的內容與方向，並且在這個時間我選擇從我熟惜的 Java 轉為 C++，原因是大份的教學資源皆為 C++ 或 Python，但因為在學習上 Java 比較容易轉換至 C++，因此在接上來的訓練皆是使用 C++ 進行。
這個月份我學習了 DFS、BFS、stack、queue

3.1.1 DFS

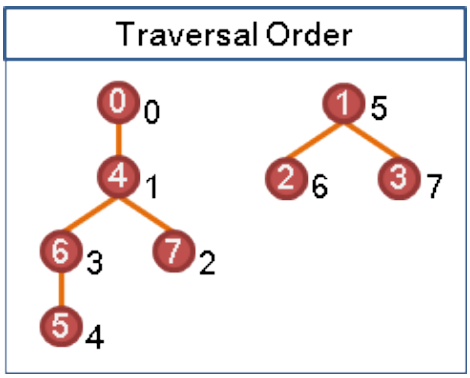
DFS（深度優先搜尋）用於遍歷一張圖，同一條路徑走到底直到沒有子結點在返回父結點。這是我在這個時間接觸比較難個資料結構。DFS 可由 stack 或遞迴實作，這裡會用到 stack 與遞迴的互換，其實就是遞迴與迭代的的互換，這也是我第一個卡也比較久的關卡。

3.1.2 BFS

BFS（廣度優先搜尋）也是用於遍歷一張圖，與 DFS 不同的是使用的資料結構不同，BFS 使用 queue 來達到搜尋的效果，會先遍歷所有當前結點的子結點，再將子結點放入 queue。



BFS 走訪順序



DFS 走訪順序

3.1.3 stack

stack 屬於先進後出的資料結構，就像抽盤子一樣，只會從最前端開始拿，也就是遞迴的原理。

3.1.4 queue

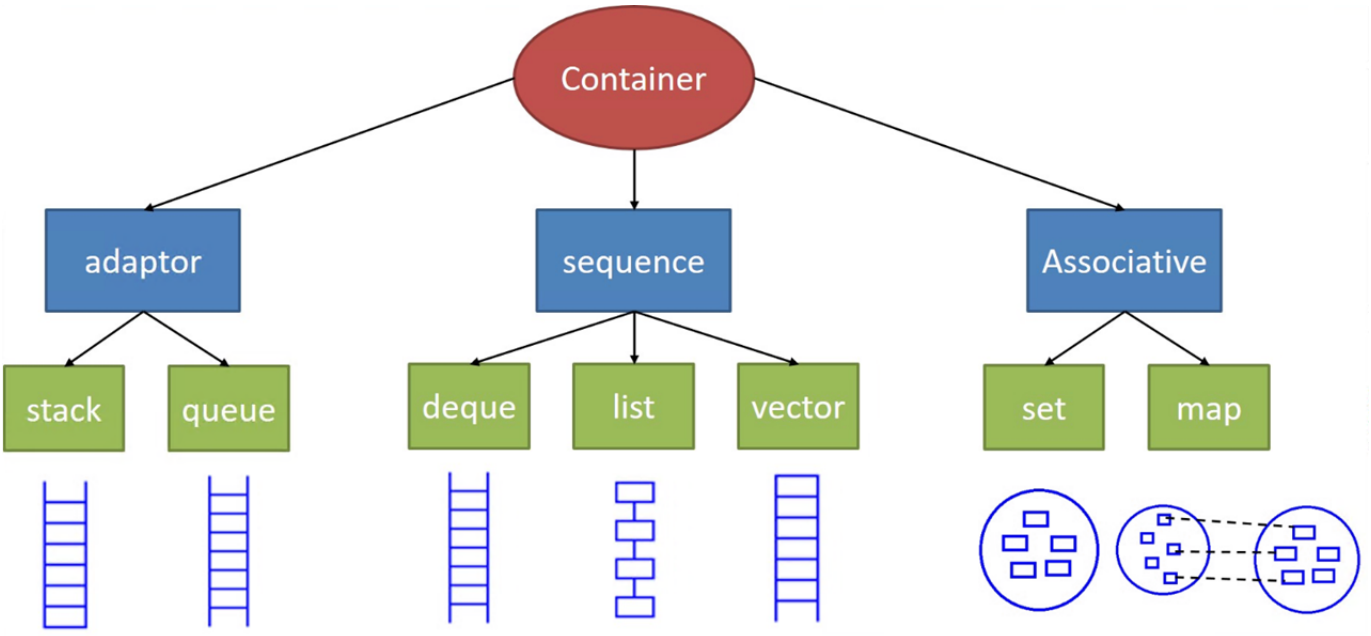
queue 屬於先進先出的資料結構，像排隊一樣，有一個頭部與尾部，常與 while 迴圈使用。

3.1.5 set

集合是非常好用的資料結構，因為其有自動排序的功能，可以和二分搜一起使用，且時間複雜度僅 $O(log_2 n)$。

3.1.6 map

map 的底層與 set 一樣都是使用紅黑樹實作，但因為 c++ 有內建，所以可以直接使用，map 是 key-value 對應的，可以快速的用 key 回應 value，就像陣列一樣，但 map 的 key 可以是任意的型別，且時間複雜度僅 $O(log_2 n)$。



3.2 第二、三月

第二個月我學習了第一次正式的課程，持續了兩個星期，這兩個星期內容包含了需多簡單和困難的內容，從演算法的時間複雜度與空間複雜度的估計，到複雜的演算法與資料結構，因此我也作了大量的題目進行練習。

課程內容包含：搜尋、枚舉、貪心、分治、數論、動態規劃、圖論、基礎資料結構、進階資料結構。

3.2.1 搜尋

搜尋的核心是用找的找到答案，因此資料常常需要排序或使用資料結構。搜尋的課程主要包含二分搜、雙指針、DFS、BFS，主要著重於二分搜的應用。

二分搜 是非常常用的基本算法之一，也是 APCS 常常當作題目的算法，主要的核心精神為將有序的資料一次又一次的砍半，實作又可分為迭代、遞迴、倍增，更可以使用 C++ 內建的函式庫。二分搜也常和建表法一起使用，可以在多筆詢問優化時間複雜度，可以達到 O($log_2 n$) 的時間複雜度。

二分搜常見的應用為對答案二分搜，意指針對給出的結果做二分搜，初始的上界與下界可以設為非常大的值。

雙指針 用於有序的資料，與 while 迴圈一起使用，當資料比預期大時就讓右指針靠向左指針，反之意然。這是常見的優化時間複雜度的方法，一般可以降低一個冪次的複雜度。

3.2.2 枚舉

枚舉是我第一個卡住的演算法，我花了比較多的時間在這上面。枚舉的精神為暴力嘗試所有的答案，雖然時間複雜度高，但有一些方法可以進行優化。枚舉有需多的方法實作，常見的像迴圈枚舉、遞迴枚舉、位元枚舉。應用包含枚舉關鍵點、折半枚舉、回朔法。

迴圈枚舉 使用像 for、while 等製作巢狀迴圈，一層枚舉一項，發現偏離目標則 break 當前的迴圈，但在實作上不比遞迴來的方便。

遞迴枚舉 遞迴的運用相當的廣，其中枚舉也其中一種，其原理很像 DFS，通常也會與回朔法一起使用，其原理為設一個起點並選擇所有可選的選項，可以使用建表紀錄所有的過程，或者是到達要的結果就直接 return。

位元枚舉 使用 bitset 或是移位符號模仿二進制，只要有'1' 就選，'0' 則捨棄，因此這個方法只能用於選或不選這種只有兩種狀態的題目，且時間複雜度為 2^n，一定要特別的注意題目的資料範圍。

枚舉關鍵點 並不是所有的情況都需有枚舉整個問題，有些時候只需有枚舉特定的點即可，通常需要使用數學化簡才能發現。

折半枚舉 假設有一筆資料的資料量高達 2^{40}，這樣若直接枚舉將會花費大量的時間，但我們可以一次枚舉一半，若一次枚舉 2^{20}，這樣的時間複雜度將會直接砍半，之後我們在將所有的資料排序，這樣即可使用搜尋的方法找出答案。

回朔法　如果今天在枚舉時答案偏差過大或無法經由後面的計算往回，則這條路徑則可以直接放棄。

3.2.3 貪心

貪心是實作容易且直覺的想法，但也因此常常造成錯誤，貪心的核心精神為只選擇當前最好的狀況，且局部問題的答案可能合成整個問題的解，如要使用貪心解題時，需要先在小部份計算，再推廣到整個問題。貪心時常需要使用數學的證明，但基礎的題目往往只需要照著直覺即可。

貪心的變化是後悔貪心，如果當前的選擇比以前還要好，則可以用陣列的方式取代以前的值。

3.2.4 分治

分治是很多常見的函式庫背後的演算法，核心精神就將大問題切成小問題直到無分割就止，實作方式以遞迴最為常見。常見的應用如排序 merge sort、quick sort 等，也用在大數的運算，如 Karatsuba 快速乘法、快速冪等，皆有使用到分治的概念。

3.2.5 數論

數學在 APCS 並不常出現，但在課程還是有提到數論，數論的內容以高中的數學為基礎出發，內容包含模運算、質因數分解、快速冪、、最大公因數等，並以程式的方法表達。

模運算　在程式設計有一個重要的運算就是模，模的意思為取餘數，因為餘數有一些重要的特性，數論的大部份都是包含了模運算。

模運算的基本性質有兩個，其中一個為數字永遠小餘模數，另一個話同餘，意指有 $(a*b)\ MOD\ c = (a\ MOD\ c)*(b\ MOD\ c)$，因此可於計算時同時取模，比起計算後在取模所需的變數範圍則可以縮小。

同餘不可用於除，如有使用除法則需先取模逆元。

質因數分解　是重要的算法之一，公私鑰密碼學 RSA 即是以大數質因數分解為基礎，給出兩個質數相乘相當容易，但要將一個數字分解回兩個質數則相當困難。分解的結果通常以數字的冪次表示。

快速冪　主要可以分成兩種實作方法，可以使用位元實作，或是把冪次拆開的方法，位元的方式較冪次拆開的方式快，但相差不會太大。

位元快速冪通常以迭代的方式實作，先新增一個變數儲存結果，再將一個冪次轉為二進制，依序走訪如果為'1' 則將結果與之相乘。

另一種實作方法則是使用分治的概念，以 $2^4 = 2^2 * 2^2$ 舉例，一個冪次可以一直不斷的拆分，直到 2^1 或 2^0，這兩者則直接返回答案，在拆分的時候如果當前的冪數為奇數則拆為 $n^{k/2} * n^{k/2} * n$，若偶數則拆分為 $n^{k/2} * n^{k/2}$，值得注意的是奇數的除 2 為無條件捨去，因此不必做額外的處理。

最大公因數　算法通常以輾轉相除法實作，這是一種非常快的算法，其時間複雜度也只有 $O(log_2 n)$，以遞迴或以迭代都可以實作，通常以遞迴實作為主，且實作量極少，，GCC 及 C++20 皆以有內建，以 GCC 為例可以使用 __gcd(n1, n2) 即可取得 n1 和 n2 的最大公因數。

在取得最大公因數之後即可透過相乘得到最小公倍數。

3.2.6 動態規劃

動態規劃 Dynamic Programming 簡稱 DP，其核心思想為將一個大問題轉由小問題先解決在回推出原問題的答案，也是經常出現的題目之一。DP 在實作時會需要一個陣列來存放計算的結果，這也是 DP 的核心，因為計算是由之前的資料所推出的，一般可分為兩種實作方法，一種為 Top-Down，另一種為 Bottom-Up。

Top-Down 又稱為記憶化，這是一種遞迴的延伸，遞迴的重點在為遞迴式，但遞迴式有一個問題是會一直計算同樣的部份，這將會大幅的增加時間複雜度，因此我們可以使用一個陣列來存放計算過的值，當檢測到該路徑以被計算過時則可直接 return 以前的計算結果，再使用這個方法之後時間複雜度會有大幅度的上升。

Bottom-Up 的計算與 Top-Down 相反，由最小的小問題開始，一步一步的往終點，但此種方法在有一些題目並不是很容易的想到，實作一樣需要一個陣列，通常會預先計算陣列的必需初始位置，計算完之後則用迭代的方式計算往後的值。

線性 DP 是最常見的 DP 型式，但也題目必需要有非常大量的練習才能比較快的反應題目的遞迴關係式，這也是線性 DP 的關鍵點，因為有了遞迴關係式就可以按照式子建立實作，迭代或遞迴都可以很好的解決此類問題，DP 的陣列不一定只有一維，題目如果有多個可能則可以用多維陣列表示。

最長共同子序列 LCS 是經典的問題之一，其目地在於在順序不變的情況下找出兩個字串的相同字元，題目所要求的解可能有兩個，一個是 LCS 的最大長度，另一個為 LCS 的字串，如只需求長度則可以在計算結束後輸出 DP 陣列的最後一個值，如還需求字串則需在計算時紀錄當前陣列的轉移來源。實作以一個二維陣列作為 DP 使用，一個維度則為一個字串的長度，時間複雜度為兩個字串的長度相乘。

背包問題 DP 初學一定會學到的演算法，背包問題並不是非常難理解的問題，主要的題意為要在一個有限重量的背包內裝入最大價值的東西，如果用枚舉解法的時間複雜度為 2^n，時間雜複度的增加是非常快速的，所以我們可以用 DP 的方式讓時間雜複度降至 n^2，實作方法為使用一個二維陣列，一個維度從最少的重量開始，另一個維度從只有一件物品開始，不斷的推算之後最後一項則為最優解。

數位 DP 有些與數字有關的問題通常是使用數學的方法解題，但有時候問題並非在為數學解，反而是每個位數的關系，題目通常也沒有辨法用數學的方式找出答案，這時即可使用數位 DP，實作依題目要作進行，可以使用 Top-Down 或 Bottom-Up，主要的重點在於要如何拆分每個位數。

滾動優化 DP 常常會用到二維陣列，也就是 n^2 的空間複雜度，如果想要降低一個維度的話就會用到滾動優化這個技巧，其原理非常的簡單，以二維陣列為例，如果要紀錄數值則可以在每一行紀錄，每一次都開新的一列，但其實新開的列是可以不需要的，因為可以使用上一次就準備好的陣列即可，且也可以使用陣列原本裡的值進行計算。

3.2.7 圖

圖是一種存資料的方式，用來表示點與點之間的關係，圖可以是單純的點的相連，但也可以是有規則的連接，單純的圖只能說是點和點的連結。

圖可以分為有向圖和無向圖，有向圖表達的意思為點到點之間的邊是有方向的且是單向的，但因為如此所以有可能會有環，也就是說從自己出發可以在回到自己，無向圖的意思只能表達點和點的連結，可以自由的來往，沒有方向性。

圖的邊常常是有權重的，權重就像過路費一樣，可以表達兩個點的關連度，也可以表達路徑的長度，權重常常是需要計算最短路徑的題型。

相鄰矩陣 是一種存圖的方式，用一個二維陣列來儲存，但此種方式其實並不那麼適合直接儲存整張圖，但用儲存權重或是一條邊通過的次數就非常的適合，因為如果有用此種方向來遍例整張圖的時間複度為 $n * m$，因此在隨機存取上沒有問題，但一旦有遍歷則會浪費很多的時間。

相鄰列表 比起相鄰矩陣更適合儲存整張圖，此種儲存方式所使用的是使用一個可以調整和插入的陣列，以 C++ 為例是使用 vector，主要核心還是二維陣列，但因為使用可以只加入特定的值，因此在遍歷的時候就不需要一個一個的走訪，因為陣列裡所存的每一個點都是可以使用的。

二分圖 指的是可以讓所有相鄰的點都是不同的顏色，因此只有兩種顏色理論上就可以就一張二分圖上色，但並非任何一張圖都是二分圖，像有環的圖就有可以非二分圖，因為有可能自己與自己相接，如此一來則該點的顏色即無法上色。

最短路徑（無權重） 是常見的圖的應用，也是常考的題型，在沒有權重的情況下我們可以使用 BFS 就可以快速的得到答案，BFS 找最短路徑不止能用在圖，也可以用在找到特定步驟的題型。BFS 使用的 queue，不能在有權重的圖運作，因為 queue 沒有排序的功能，在有權重的圖則要使用特定的演算法。

最短路徑（有權重） 大部分的帶權重的圖都是以正數為例，但以 APCS 的題目其實並不常考此類題目。帶權重的最短路徑演算法其實和無權重的計算類似，但 queue 需要改為 priority_queue，原因為 queue 只能記錄每個點但不能以權重來排序，因此選用 priority_queue 則可以在 BFS 的同時做到權重排序，以最小的優先以此往外，且 priority_queue 的時間複雜度僅 O($log_2 n$)，這樣的算法被稱為 Dijkstra Algorithm，但有一點要注意的是不可用於路徑為負的狀況，否則計算上將會出現問題。

3.2.8 前綴和與差分

假設有一個陣列要計算特定段的總合，且有多筆的查詢，此時每次都計算的話會浪費大量的時間，因此可以預先計算從頭到尾的和，再由終點減去啟點即可算出啟點到終點的所有合。前綴和也可以用於二維陣列，但在建立上比較麻煩，使用排容定理使重複的值不會被重複計算

差分的意思為兩兩數值的差，這樣的資料結構有一個很好用的特性，在差分的某點加上特定的值，在另一個點在減去等定的值，如此當計算總合時即可為整個區間加值，大幅優化時間複雜度。

3.2.9 BIT 樹

BIT 樹在一些特定的問題相當的好用，主要功能為可以隨時增加特定區間的值，這個功能是前綴和所沒有的，而且也可以像差分那樣做到區間加值，實作也相當的簡單，大致的實作過程是用 lowbit 放進 for 迴圈裡做

倍增法，雖然 APCS 的題目並不會有需要自己實作資料結構的題目，但有些題也可以使用特定的資料結構就題目變得簡單，。

3.2.10 線段樹

線段樹和的功能大至相同，但有一點則是 BIT 樹所無法取帶的，線段樹有一個重點在於區間加值，實作線段樹以遞迴為核心，因此線段樹可以同時查詢非常多的東西，像最大區問和問題，如果同時有加值則只能用線段樹實作，這也是線段樹無法被取帶的理由。

3.2.11 並查集

有些題目有非常多但不連同的圖，或者要處理關連性問題都很適合用並查集，因為其可以快速的查詢各資料之間的關係，平均時間複雜度最快可以達到 O(1)，因為在建立的時候可以使用路徑壓縮的技巧，實作也相當的簡單且快速。

3.3 第四、五月

最後兩個月是衝刺期，我找到了中正大學 吳邦一教授 的講議 AP325 並進行練習，這分講議補足了我之前所沒有學會的算法，也讓我學會了可以在 APCS 實際使用的技巧。

3.3.1 遞迴

這是最基本的結構，但我並沒有到非常的精熟，因此我還是在一些題目卡住了。例如有一些題目的測資需使用遞迴以讀取，也有可以是解法本身就是以遞迴為主軸，更有題目的測試資料即為 DFS 的資料，這些都是在我之前所不知道的。

3.3.2 滑動窗口

滑動窗口其實是雙指針個運用，適合用於有一個大區間但要求其中包含的小區間的特值，實作方式可以用 map 或者 deque，同樣也可以使用離散化等技巧，且時間複雜度只有 O(n) 或 O($log_2 n$)。

3.3.3 掃描線

掃描線常常很 map 一起使用，因為 map 可以使用迭代器的方式遍歷，如此一來就不需要從頭開始一步一步的檢查，其原理為在開始特定的點增加數值，然後在要結束的點減去特定的數值，之後在要計算答案的時候只要重頭跑一次即可，時間複雜度大約為 O($log_2 n$)。

3.3.4 set、map

這兩個容器我在之前就有學過了，但我並不知道有什麼樣的變化，這裡所學習到的技巧為離散化，離散化其實原理簡單，即為將所有的測資以名次排序，且計算也只使用其名次，實作相當容易步驟為排序 -> 去除重複的值 -> 以原本的陣列對名次二分搜，這樣的算法非常的實目，可以省下非常多的空間。

3.3.5 bitset

位元的運算並不常出現，但其實很多的問題都可以使用到此資料結構，因為 bitset 可以直接使用位元運算，且速度非常之快，其原理為布林陣列，因其特性所以非常適合像需要取交集、差集等題目。

3.3.6 stack

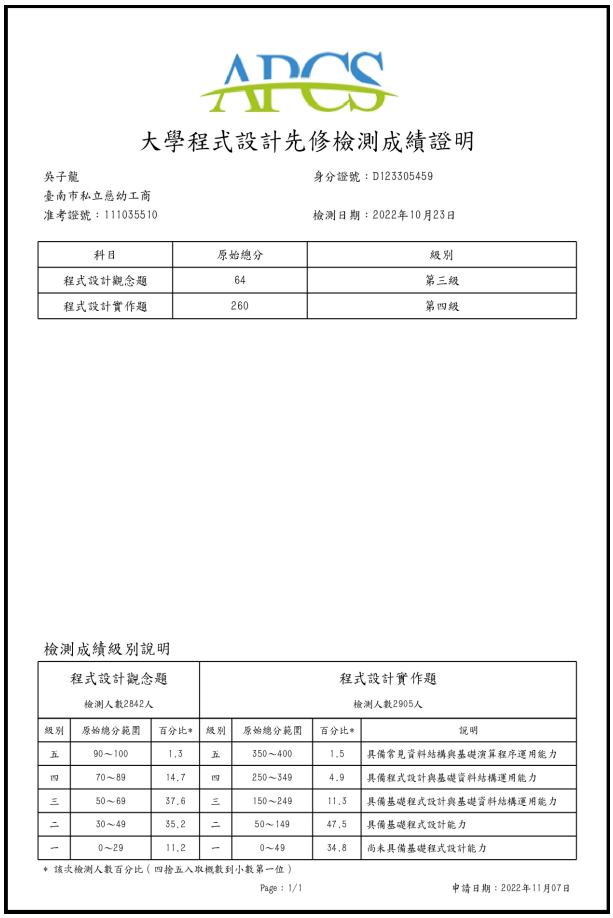
這個資料結構簡單且好使用，因此也是 APCS 常出現的題型，但也有題目需要使用較複雜的技巧，常見像有單調棧，單調性是非常好用的特性但也比較困難，以單調棧為例所需要維護一個由小到大或由大到小的 stack，但因為 stack 並不能很好的取得內部的值，所以常常要和 map 一起使用，刪除的時候和 map 一起刪除，增加的時候和 map 一起增加。

3.3.7 deque

deque 其實核心即是一個由指標串成的 linkedlist，因此特性基本上完全的一樣，deque 的常見運用為單調隊列，這與使用 stack 的單調棧很像但加入了時間性，因為 deque 是雙頭皆可以插入與刪除，所以可以在尾部刪去過期的資料，如此一來便能容易的計算有限定範圍的題目。

4 結論

4.1 分數



大學程式設計先修檢測成績證明

吳子龍
臺南市私立慈幼工商
准考證號：111035510

身分證號：D123305459
檢測日期：2022年10月23日

科目	原始總分	級別
程式設計觀念題	64	第三級
程式設計實作題	260	第四級

檢測成績級別說明

程式設計觀念題 檢測人數2842人			程式設計實作題 檢測人數2905人			
級別	原始總分範圍	百分比*	級別	原始總分範圍	百分比*	說明
五	90～100	1.3	五	350～400	1.5	具備常見資料結構與基礎演算程序運用能力
四	70～89	14.7	四	250～349	4.9	具備程式設計與基礎資料結構運用能力
三	50～69	37.6	三	150～249	11.3	具備基礎程式設計與基礎資料結構運用能力
二	30～49	35.2	二	50～149	47.5	具備基礎程式設計能力
一	0～29	11.2	一	0～49	34.8	尚未具備基礎程式設計能力

* 該次檢測人數百分比（四捨五入取概數到小數第一位）

Page：1/1

申請日期：2022年11月07日

4.2 檢討

這一次 APCS 的分數我其實還算滿意，但每次的考試都一定有沒有表現好的點，對於 APCS 的評分方式一直是考試的難點，雖然我知道這一點，但因為我在考試時過於緊張，因此我在考試的時間分配不好，我原本給自己的安排是 1->3->4->2 的順序，前兩個題目照我所想的很快的解出題目，但因為我在第四題的時候我所想到的解法其實是正確的，但因為我沒有實作，所以我浪費了一些時間在這個上面，所以我直接看第二題，但其實這兩題我都沒有到非常有把握，但我還是作了一下第四題但實作是錯誤的，所以我就直接往第二題，但因為我浪費了過多的時間，所以我的第二題實作並不漂亮，也沒有時間慢慢的測試，因此我覺得這一點必需要調整，要冷靜的處理狀況。

我在準備的時間發了很多在實作題上面，但我的觀念題僅有把程式設計實習的考古題進行練習，但考試有很多的演算法題目，我發了很多的時間，而且題目也比我想像的難很多，有一些題目是要畫出遞迴樹，但因為我沒有事先的練習，因此解題花費的時間也是最多的，這方面我需要在加強。

4.3 心得

最初我以為 APCS 的題目只是基本的語法而以，但在於深入看了考古題之後發現有很多的題目我都是解不開的，而且我以前是以 Java 為核心學習基本的資料結構，因此我還是花了滿多的時間從 Java 轉到 C++，一切都要重新學習，而且我一直以為時間是不夠的，但我還是堅持不停的學習，也還好我在暑假的時候可以參加到第一次正式的課程，但我沒有什麼基礎，所以學起來特別的吃力，但就算如此我還是花了非常多的時候在練習題目，一步一步的往前進步，後面的兩個月雖然學習的難度沒有暑假那麼難，但這也才讓我知道我有什麼不足，重新複習以前所學的，這一次考試是我最緊張的一次，考試那天我也是用了全力才考到我理想的分數，這四年的程式之路終於有了一個認證。程式的路還有很長，不止程式還有電腦相關的種種都是我要學習的，這只是一個開始而以，未來一定還有更多的挑戰等著我。