資料結構

1. 結構 structures

   假設我們今天要用程式列印出一位學生此次段考成績，我們會宣告一組一組變數，使用 char 宣告姓名、int 宣告學號，float 宣告此次段考每一科考試成績，

   char name[7]；char id；float score；int rate；

   但分開宣告不方便管理，此時可使用 structure 解決。定義一個 struct student，裡面放入這位學生的姓名、學號及此次段考每一科成績。

   struct student{
   char name[7]；
   char id；
   float score；
   int rate；
   }student1

   Example:

```
#include <iostream>
using namespace std;
struct student {
char name[20];
char id[7];
float score;
int rate;
}student1;
int main() {
cin >> student1.name;
cin >> student1.id;
cin >> student1.score;
cin >> student1.rate;
cout << student1.name << endl;
cout << student1.id << endl;
cout << student1.score << endl;
cout << student1.rate << endl;
}
```

A. 線性結構

(1) 線性結構 ： 數據元素之間存在一對一的線性關係

(2) 線性結構常見的有： 陣列、佇列、堆疊 (stack)

B. 非線性結構

二維陣列、多維陣列

---

2. 指標 pointers (有點難，再多找點資料!!) (多補充一點)

https://www.youtube.com/watch?v=MgR5x1jdvJs

https://kopu.chat/c%E8%AA%9E%E8%A8%80-%E8%B6%85%E5%A5%BD%E6%87%82%E7%9A%84%E6%8C%87%E6%A8%99%EF%BC%8C%E5%88%9D%E5%AD%B8%E8%80%85%E8%AB%8B%E9%80%B2%EF%BD%9E/

https://opensourcedoc.com/c-programming/pointer/

https://hackmd.io/@howkii-studio/Bkf-2DQiw/https%3A%2F%2Fhackmd.io%2F%40howkii-studio%2Fdata_structure

指標(pointer)是一種特殊的變數，用來存放變數在記憶體中的位址。當我們宣告一個變數時，編譯器便會配置一塊足夠儲存這個變數的記憶體給它。每個記憶體空間均有它獨一無二的編號，這些編號稱為記憶體的位址(address)，程式便是利用記憶體的位址來存取變數的內容。位址有如住家的門牌號碼，在程式中是獨一無二的。系統可以依位址來存取變數，就如同郵差可以依門牌號碼來送達信件一樣!

---

3. Linked list 鏈結串列

(1) 記憶體位置不連續，以隨機的方式儲存

(2) 各節點的資料型態不必一定相同

(3) 每個點放在不同記憶體位置，不會按線性的順序儲存資料

(4) 記憶體非連續，不需事先知道整體資料大小

(5) 每一個節點裡存有到下一個節點記憶體位置的 pointer

(6) 可以有單向或是雙向的 linked list

(7) 能夠被直接存取的節點只有最前面的第一節點

(8) 因為不用事先定義好一塊連續的記憶體空間，所以插入或刪除資料都很方便

(9) 當想查詢特定節點(node)時，必須從頭節點開始走訪

---

4. Array 陣列
   (1) 通常用來儲存有序列的相同資料於連續記憶空間
   (2) 一個 array 會分配一塊連續的記憶體來儲存
   (3) 必須預先知道整體資料大小來分配記憶體
   (4) 利用元素的索引(index) 可以計算出該元素對應的儲存位址
   (5) 可以有一維與多維的陣列
   (6) 可以透過 index 直接存取各個元素的值
   (7) 須事先宣告固定的記憶體空間，因此容易造成記憶體浪費
   (8) 讀取與修改串列的資料時間是很快的
   (9) 刪除或加入新元素需要移動大量資料

Ps : 準備一下 array 跟 linked list 的比較

Array 跟 Linkedlist 比較

Linked List 跟 Array 都可以用來儲存資料，但在使用情境不同的情況下，都有各自不同的好處與壞處

(1) Array 是由相同類型的元素的集合所組成的資料結構；Linked List 各節點的資料型態不必一定相同。
(2) Array 必須先知道整體資料大小來分配記憶體；Linked List 記憶體不是連續的，不需事先知道整體資料大小
(3) Array 在讀取與修改資料時間是很快的，可使用 index 的方式來讀取資料；Linked List 則必須從頭節點開始走訪 ex: node1.next.next.next

---

5. Stack
   (1) 比喻成像書一本本往上堆
   (2) 資料一點點限制
   (3) 按照順序拿: 先進後出 First in last out (FILO)
   (4) push pop peek
   (5) 生活用途: 上一頁
   (6) call stack (程式呼叫)

什麼是 stack?

堆疊遵循著資料先進後出(First in last out) (FILO)，先進去後出的原則。。常見的兩種操作方式，分別是 push 與 pop，push 就是把東西放到最上面，pop 則是把東西從上面拿走。生活中常見的例子有: 自助餐吃到飽的盤子、總是被媽媽塞得滿滿，深不見底的冷凍庫等等。

什麼是 queue?

佇列遵循著資料先進先出 (First in first out) (FIFO)，第一個進去，第一個出來的原則。常見的兩種操作方式，分別是 enqueue 與 dequeue。Enqueue 是丟資料到 queue 內，dequeue 是從 queue 內將資料取出。生活中常見的例子有: 排在高速公路出口閘道的車流、打電話給客服，滿線時須等待。

Stack 跟 Queue 的比較

1. 資料處理的順序: Stack 是先進後出。Queue 是先進先出
2. 新增/刪除的端點:
   (a) Stack 只有一個端點，新增和刪除在同一端點，依序向上堆疊，刪除會從最上面先刪除。
   (b) Queue 有兩個端點，新增會從佇列尾端放入，刪除則從最前端的舊資料先刪除
3. 常見應用:
   (a) Stack : 像是平常很常使用的編輯器 word，回上一個步驟這個指令
   (b) Queue: 安排多個程式的執行順序

---

6. Queue
   (1) 排隊
   (2) 先進先出 First in first out (FIFO)
   (3) enqueue、dequeue、peek
   (4) 生活用途: 搶票、打電話給客服，滿線時須等待

https://justnote.coderbridge.io/2022/03/06/stack-and-queue/

---

7. Binary tree https://www.youtube.com/watch?v=xDwFMffqLbw

樹有幾個特性:

(A) 不包含迴路 (比喻成粽子)

(B) 一棵樹中的任意兩個節點有且僅唯一的一條路徑連通

(C) 一棵樹如果有 N 個節點，那一定恰好有 N-1 條邊

(D) 在一棵樹中加一條邊將會構成一個迴路

(E) 名詞解釋:

(a) 節點 (node)

(b) 根(root)

(c) 葉節點 (leaf)

(d) 父 (parent)

(e) 子 (child)

(f) 兄弟 (siblings)

Tree : (還沒完成!!)

https://medium.com/%E6%8A%80%E8%A1%93%E7%AD%86%E8%A8%98/%E5%9F%BA%E7%A4%8E%E6%BC%94%E7%AE%97%E6%B3%95%E7%B3%BB%E5%88%97-tree-%E6%A8%B9%E7%8B%80%E8%B3%87%E6%96%99%E7%B5%90%E6%A7%8B-d10fe8ac1ce2

(1) 特性 :

（ａ）由一個根節點 (root)與多個子節點 (child node) 所組成

（ｂ）每個 node 的節點數量稱做 degree，可以有多個子節點

（ｃ）每個 node 會記錄他的子節點是誰與在節點上儲存的資料

（ｄ）每個 node 只能有一個父節點

（ｅ）每一條分支都可以看作一條 linked list

（ｆ）沒有子節點的 node 又稱作為 leaf node

（ｇ）樹裡面沒有環路 (cycle)

(2) 如果把剛剛的 tree，限制每一個 node 最多只能有兩個子節點，這時候就可以稱作為 binary tree 二元樹

(a) 特性 :

(b) 每一個 node 最多只能有兩個子節點

(c) 子節點有左右之分 (left node, right node)

(3) 如果再把訂定更多條件，他會成為一個方便搜尋的 binary search tree (二元搜尋樹/有序二元樹)

(a) 特性:

(b) 左子樹上所有節點的值均小於它的根節點的值
(c) 右子樹上所有節點的值均大於它的根節點的值
(d) 任意節點的左、右子樹也分別為二元搜尋樹
(e) 不會出現有重複值的節點

二元樹有幾種遍歷:
(1) Pre-order 前序: root-left-right
(2) In-order 中序: left-root-right
(3) Post-order 後序: left-right-root
(4) Level-order 層序: 一層一層從左至右

DFS (Depth-First Search) 深度優先搜尋
核心邏輯: 如同 Pre-order，先往第一個節點的最深處走，再去找相鄰的點，如果該節點的邊上每一個節點都走過，則回到上一個點，繼續尋找其他還沒走過的點

實作: 通常可以用比較直覺的遞迴方式，或是利用 stack 的 first-in-last-out 特性來達成

Breath-First Search (BFS) 廣度優先搜尋
使用 LEVL-ORDER 層序搜尋的方式，把每一層的所有節點從左到右都走過一遍，才會往下一層接著去做一樣的邏輯，直到所有的節點都走完

---

8. Sorting (排序意思是從小排到大，或從大排到小!)
https://medium.com/%E6%8A%80%E8%A1%93%E7%AD%86%E8%A8%98/%E5%9F%BA%E7%A4%8E%E6%BC%94%E7%AE%97%E6%B3%95%E7%B3%BB%E5%88%97-%E4%BD%A0%E5%8F%AA%E6%9C%83%E7%94%A8-built-in-function-%E4%BE%86%E6%8E%92%E5%BA%8F%E5%97%8E-4a196d37f9f5
sort 種類
(1) 氣泡排序 (Bubble sort)
(A) 想像一下，像是氣泡一樣不斷往上升，一次比較兩個元素，如果它們的順序錯誤就把他們交換過來，每次都會將最大值升到最頂端
(2) 選擇排序 (Selection sort)
(A) 找出一個最小的
(B) 找出來 N，找 N 次
(C) 每次從 i 到 n 筆中挑出最小值，和前面第 i 筆交換

Example : [2,5,1,7]

(1) 先從 n0 開始做到 n，找到最小值 1，把他跟第 0(i) 個交換
[1,5,2,7]

(2) 接著從 n1 開始做到 n，找到最小值 2，把它跟第 1(i)個交換
[1,2,5,7]

(3) 接著重複做到最後一個為止，即完成排序

(3) 插入排序 (Insertion sort)

(A) 找到適當的位置插入進去

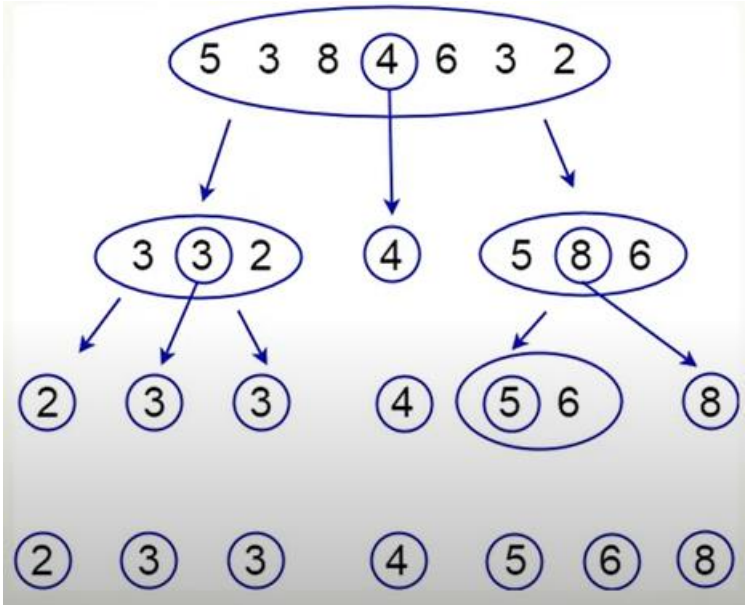
(B) 核心概念: 把第 i 筆資料插入前面(i-1)筆已經排序好的陣列中

PS: 氣泡排序、選擇排序、插入排序相較簡單，面試比較不會問

(4) 快速排序 (Quick sort) (面試愛考!!)

(A) 將一個難以直接解決的大問題，分割成一些規模較小的相同問題，以便各個擊破，分而治之 (Divide and conquer)

(B) Example:



演算法文字描述如下:

(1) 找出一個 pivot

(2) 把比 pivot 小的值放 pivot 左邊

(3) 把比 pivot 大的值放 pivot 右邊

(4) 數列根據 pivot 切分為兩個數列，兩個數列繼續往下一樣的動作

(5) 合併排序 (Merge sort) (面試愛考!!) (要再多找資料!!)

(A) 也是 Divide and Conquer 演算法的一種，主要核心的概念，就是先分割再合併

快速排序 面試比較會問

合併排序 面試比較會問

---

9. Graph https://ithelp.ithome.com.tw/articles/10208277

在資料結構上指的是點和點之間的關聯的東西

(1) 有向圖: 邊有方向性，表示兩點之間為單向關係

(2) 無向圖: 邊無方向性，表示兩點之間為雙向關係

(3) Eagle List: 用陣列的方式，記錄點與點之間的邊

(4) Adjacency Matrix: 把一張圖上的點依序標示編號，然後建立一個矩陣，來記錄連接資訊。方陣中的每一個元素都代表著某兩點的連接資訊。

(5) 深度優先搜尋(DFS)
https://www.youtube.com/watch?v=c4R27mPT5kY

(6) 廣度優先搜尋(BFS)
https://www.youtube.com/watch?v=c4R27mPT5kY&t=496s

Graph
https://medium.com/%E6%8A%80%E8%A1%93%E7%AD%86%E8%A8%98/%E5%9F%BA%E7%A4%8E%E6%BC%94%E7%AE%97%E6%B3%95%E7%B3%BB%E5%88%97-graph-%E8%B3%87%E6%96%99%E7%B5%90%E6%A7%8B%E8%88%87dijkstras-algorithm-6134f62c1fc2

(1) 定義中會由點與線，來描繪出一個 Graph

(2) 如果兩點間的線有著方向性的關係，會使用箭頭來表示方向性，稱有像圖；反之則為無向圖，同時具備兩種的圖稱為混合圖

(3) 可以用 adjacency matrix(相鄰矩陣)跟 adjacency list(相鄰串列)來表示

(4) Adjacency matrix (相鄰矩陣) : 使用二維陣列，有關聯的點之間為 1

(5) Adjacency List (相鄰串列): 使用 Linked-List，鏈結的點順序不重要

10. Hashing　(有點難!!!!) https://pjchender.blogspot.com/2020/05/hash-table.html
https://www.youtube.com/watch?v=vPvxEDyxI2w

(1) 又稱哈希、切細、雜湊、散列(中國)

雜湊表

輸入變成一個值

一個 key 找到一個 element

Ps: 如果遇到一個 key 有兩個資料的話，使用雜湊衝突來解決，方法有:

(1) Closed addressing (像 linked list) : 接在後面

(2) Open addressing : 往下找，找到空位填進去。

舉例: 查找電話簿中某人的號碼，可以建一個按照人名首字母順序排列的表。當我今天要找「王」姓的電話號碼，可以直接在首字母為 W 的表中找到。速度上比從電話簿第一頁從頭找起還來的快速許多。
