中原大學資訊工程系 演算法分析第二次機測

Deadline: 6 / 13 / 2025 (星期五)
(限期末考前測完，逾期不得補繳)

【程式設計說明】

1. 每組限 2～3 人，組員須固定，本學期不得任意變更。原則上以專題組員為主。
2. 組員應合作共同解題，但嚴禁跨組合作。
3. 程式設計必須使用 Python 程式語言，版本請採用目前最新版本 (原則上，請直接下載與安裝 Anaconda)。
4. 可參考課本、參考書籍或網站資料等進行解題，解題方法及演算法不限，但嚴禁抄襲他組程式，組員均有責任保護程式不被他組抄襲。**若發現抄襲屬實，兩組均以零分計。**
5. 輸入與輸出採用標準格式或讀寫檔案方式進行。
6. 每一支程式均須附上組員姓名及學號，例如:

```
#    演算法分析機測
#    學號: 11127XXX / 11127XXX
#    姓名: 陳○○ / 林○○
#    中原大學資訊工程系
```

程式命名依該組學號在前的同學【學號＋題號】為原則。例如:
11127001_1.py
11127001_2.py
…
11127001_5.py

【機測須知】

1. 評分以解題成功之題數多寡與執行時間決定。
2. 程式必須能處理不同的輸入資料 (但輸入格式與範例相同)，並輸出正確結果 (輸出格式必須與範例相同)，組員應能說明程式設計內容，方可視為成功。程式的輸出結果錯誤、輸出格式與範例不符、或在執行後超過 5 秒 (以每筆測資為基準) 仍未結束，均視為失敗。若程式測試失敗給予基本分數，未繳交程式則以零分計。
3. 本機測於規定之期限前，各組應攜帶程式原始碼至電學大樓 603 室找助教測試 (電話：265-4726)，每組限繳交一次，不可分題或多版本繳交，逾期不得補繳。本次機測程式碼須於 Deadline 前上傳至 i-learning【作業區】。
4. 助教將使用不同之輸入資料作為測試與評分依據，同學應在繳交前充分測試程式。
5. 機測成績納入學期平時成績計算，請同學把握。

指導教授: 張元翔

【執行時間測試】

機測預計採用個人電腦 CPU Intel i7、8G RAM、作業系統以 Windows 10 為主。建議同學在繳交程式前先使用下列 Python 程式進行初步的執行時間測試：

```
import time
start_time = time.time()
……
total_time = time.time() – start_time
print(total_time)
```

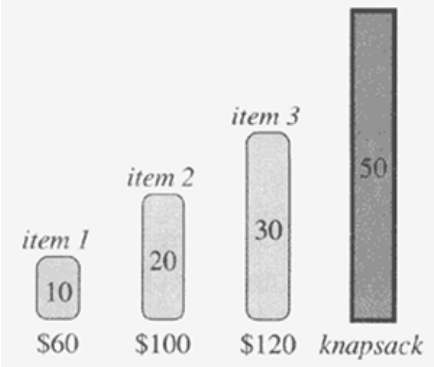
I. 0-1 背包問題 (0-1 Knapsack)

0-1 Knapsack 問題 (或稱為 Bin-Packing 問題) 是電腦演算法中具有代表性的問題，問題描述如下:

有一小偷到一家商店偷東西，他發現 *n* 項物品，每項物品各有不同價值及不同重量，小偷的目的是帶走總價值最高的物品，但他能帶走的**背包** (Knapsack) 有重量的限制。試設計程式解決 0-1 背包問題 (即每項物品僅能**帶走**或**不帶走**，無法帶走部分)，並須求得**最佳解** (Optimal Solution)。

輸入說明:

輸入物品 Knapsack 重量 *W* 與物品總數 *n*，接著分別是各項物品的重量與價值 (均為正整數，中間以空格隔開)。以下為輸入範例:



輸出說明:

求出可能的最高總價值，並列出帶走物件的編號 (須按編號由小到大順序排列，並以逗號隔開，最後的物件不能有逗號)。

輸入範例:

50
3
10 60
20 100
30 120

輸出範例:

Total Value = 220
Take Items 2, 3

II. 霍夫曼碼 (Huffman Codes)

霍夫曼碼在**資料壓縮** (Data Compression) 中是常見的技術之一，被廣泛使用於文件、音訊、影像、視訊等多媒體壓縮應用中。霍夫曼碼的主要原理是由於表示資料的方式可以分成兩種：若使用**固定長度碼** (Fixed-Length Codeword)，則每一個字元是採用固定長度的編碼方式；霍夫曼碼是比固定長度編碼更為有效的編碼方式，採用**可變長度編碼** (Variable-Length Codeword)，使得所需的位元數大幅降低。

以下述字元編碼為例，試參考課本 (講義) 描述的演算法，設計 Python 程式完成霍夫曼碼的**編碼** (Encoding) 及**解碼** (Decoding)。

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

【註】本機測不可使用第三方開發的 Python 軟體套件。

輸入說明:
每組輸入包含的字元數 *n* (均為正整數)，0 表示結束，緊接為每一個字元及其發生頻率，所有字元均可能是英文字母大或小寫，且頻率均為正整數 (但不會事先排序)。最後，給定一個特定的二元碼，試使用霍夫曼碼對其進行解碼。

輸出說明:
就每組輸入列出結果，包含：(1) 每一個字元的霍夫曼碼；及(2) 解碼之結果。

輸入範例:
6
a 45
b 13
c 12
d 16
e 9
f 5
01001101
6
A 2

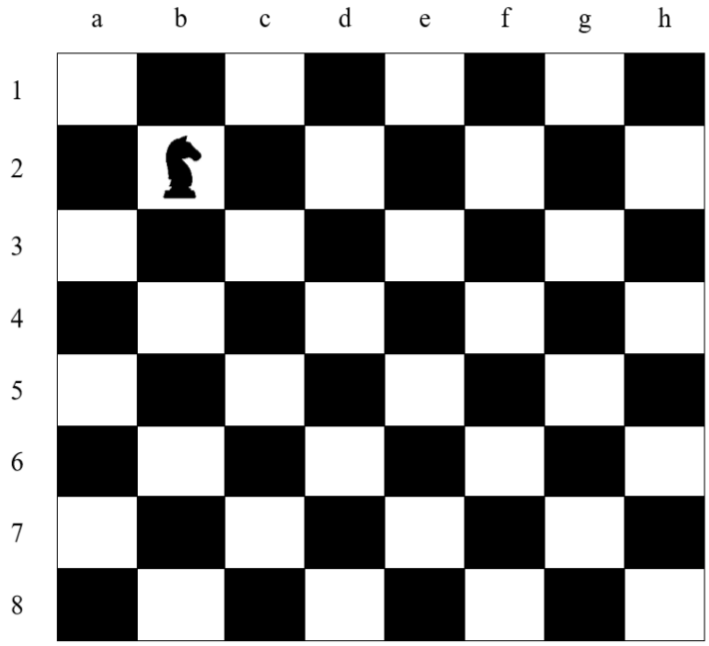
B 6
C 15
D 12
E 8
F 3
010101001100
0

輸出範例:
Huffman Codes #1
a 0
b 101
c 100
d 111
e 1101
f 1100
Decode = ace

Huffman Codes #2
A 0100
B 011
C 11
D 10
E 00
F 0101
Decode = FACE

III. 西洋棋騎士 (Chess Knight)

西洋棋 (Chess) 是一種二人對弈的戰術棋盤遊戲，也是世界上最流行的遊戲之一。本問題中，將探討西洋棋**騎士** (Knight) 的移動，如下圖。西洋棋盤是一個 8 × 8 的棋盤，每一列使用 1 ~ 8 編號；每一行則使用 a ~ h 編號。



我們想要解決的問題是：「給定兩個位置 X 與 Y，若騎士從 X 到 Y 至少需要走幾步?」

舉例說明，若想將騎士從 b2 移到 c3，至少需要 2 步，即先將騎士從 b2 移到 d1，再從 d1 移到 c3。另一種走法，是先將騎士從 b2 移到 a4，再從 a4 移到 c3，但這兩種走法的移動步數相同。

輸入說明:
兩個西洋棋的座標位置。每個座標位置是由一個小寫英文字母 (a ~ h) 與一個數字 (1 ~ 8) 組成；0 0 代表結束。

輸出說明:
騎士至少需移動的次數。
【註】答案必須是最少次數。

輸入範例:
b2 c3

a1 b2
a1 h8
0 0

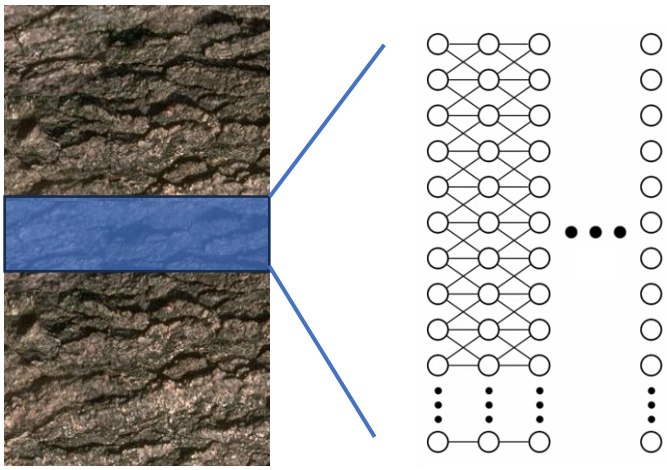
輸出範例:

From b2 to c3, Knight Moves = 2
From a1 to b2, Knight Moves = 4
From a1 to h8, Knight Moves = 6

IV. 紋理拚貼 (Texture Stitching)

影像處理領域中，**紋理拼貼** (Texture Stitching) 是一種將多張數位影像 (通常是部分紋理影像) 無縫拼貼成一張較大尺寸的影像處理技術。以下列紋理影像為例，若直接採取上下 (垂直) 方向拼貼，則接縫處明顯不連續，使得拼貼效果不太理想。

若使用**動態規劃法** (Dynamic Programming)，可以達到「無縫拼貼」的效果。首先，設定影像重疊比例，則進行兩張數位影像的上下 (水平) 方向拼貼，如下圖。首先設定重疊比例介於 20%～30%之間，例如：20%。假設原始影像為 256 × 256，則重疊區域的高是 256 × 20% = 51 (四捨五入取整數)。



因此，可以將重疊區域視為一個網路，網路架構如同演算法中介紹的**組裝線排程** (Assembly-Line Scheduling) 問題。每個像素 (節點) 根據 R、G、B 值的歐氏距離計算，即：

$$\sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$

用來評估重疊區域像素間的相似度，其中 (R_1, G_1, B_1)、(R_2, G_2, B_2) 分別為上、下影像的 R、G、B 值。此外，每個節點連接至上、中、下三個節點。

參考 Assembly-Line Scheduling 的演算法，求得的最短路徑，即是理想的「接縫」(Seam)。拼貼時若像素落在接縫上方，則採用上方影像的像素；若落在接縫下方，則採用下方影像的像素。下圖為垂直拼貼範例，形成「無縫拼貼」的效果。

【註】原則上，「接縫」採用上方影像的像素。

相關技術說明如下:

- 安裝 OpenCV 進行數位影像檔的讀寫。

```
pip install opencv-python
```

- 典型的 Python 程式範例如下：

```
import numpy as np
import cv2

img = cv2.imread("Texture_Rock.bmp", -1)
cv2.imshow("Example", img)
cv2.waitKey("0")
```

- 彩色影像是以 3 維陣列存取，資料型態為 uint8 (每個像素包含 3 個 Bytes，分別表示 B、G、R 三原色。每個 B、G、R 的色彩值是以 1 個 Byte 表示，介於 0～255 之間。

輸入說明:
採讀檔方式進行，並輸入相關參數。原則上，輸入的紋理影像大小為 256 × 256 像素。

輸出說明:
輸出拚貼好的結果影像檔。

輸入範例:
請輸入影像檔：Texture_Rock.bmp [Enter]
請輸入拼貼方向 (1)水平、(2)垂直：2 [Enter]
請輸入重疊比例 (%)：20 [Enter]

輸出範例:
輸出影像檔 Texture_Rock_result.bmp
【註】原圖檔案名稱_result.bmp。

V. 拼圖 (Puzzle)

拼圖 (Puzzle) 遊戲是常見的益智遊戲。本機測嘗試使用電腦程式設計，自動解決拼圖問題。典型的拼圖如下：

海賊王 (One Piece) 原始影像

海賊王 (One Piece) 拼圖

拼圖遊戲說明如下：

- 原始影像為彩色影像，影像大小為 1920 × 1080 像素 (寬×高)。
- 每張小拼圖塊的大小為 120 × 120 像素，因此總共有 16 × 9 張小拼圖塊。
- 每張小拼圖塊僅在幾何空間中進行任意平移，並無旋轉或其他操作。任意平移後進行拼接，次序則以隨機方式進行。

解題方式提示如下：

- **節點** (Nodes)：每一張小拼圖塊視為一個節點。
- **邊** (Edges)：每一條邊連接一對拼圖塊，邊的權重表示這兩塊邊緣的相似度 (例如：邊緣像素的歐氏距離)。
- **圖形** (Graph)：根據輸入影像建構圖形結構。
- **最小生成樹** (MST)：使用 Kruskal 或 Prim 演算法，找出連接所有小拼圖塊且權重總和最小的生成樹，代表小拼圖塊最可能的相鄰方式。
- **走訪** (Traversal)：走訪最小生成樹，以決定每個小拼圖塊的相對位置，進而重建原始影像。

試設計程式讀取拼圖之影像檔，例如：One_Piece1.bmp，輸出拚好的原始影像，檔案名稱為 One_Piece1_result.bmp。

【註】本問題的計算時間限制為 1 分鐘 (使用 Intel Core i7 CPU)。

輸入說明:
採讀檔方式進行。

輸出說明:
輸出拚好的影像檔，影像檔案格式採用.bmp。

輸入範例:
請輸入影像檔: One_Piece1.bmp [Enter]

輸出範例:
輸出影像檔: One_Piece1_result.bmp
【**註**】原圖檔案名稱_result.bmp。