Computational Thinking & Programming
資料壓縮技術 Data Compression

Concepts: Data presentation, data compression

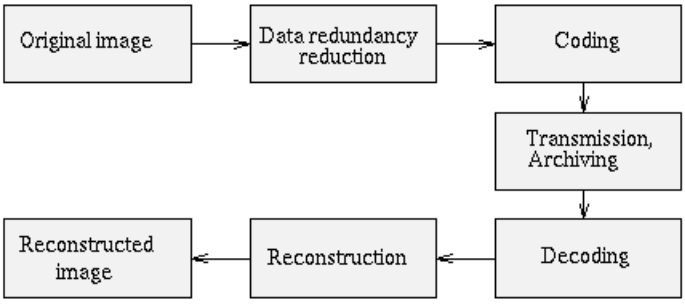


Figure 12.1 *Data compression and image reconstruction.*

資料壓縮是透過編碼的技術，來降低資料儲存時所需的空間，等到我們要用時，再做解壓縮的動作即可。

資料經過壓縮後，除了需要較少的儲存空間外，當我們在網路上傳輸時，所需的傳輸時間也較短。因此，我們從網際網路下載的資料，常常是壓縮後的資料，這樣我們才能更快速取得資料，在解壓縮後，我們就能還原成本來的資料。

我們如何做資料壓縮呢？下面這些技巧是常常被使用的。我們可以根據資料內各個字符出現機率的不同，來決定表示該字符所對應的二元碼(0 與 1 位元的組合)長度，我們用較短的碼來表示出現機率較高的字符，用較長的碼來表示出現機率較低的字符，這樣平均而言，我們所需的位元數會比我們用等長的碼來表示每個字符的情況來得省。例如在英文中，字母 E 是出現機率最高的字符，而字母 Z 是出現機率最低的字符，所以比較好的編碼方式是用最短的碼來表示 E，而用最長的碼來表示 Z。

另一種壓縮技巧是將重複性的資料以它們的特質來表示。例如，如果我們的資料內容為 111…，總共有一千個 1，與其用一千個位元來儲存這些都是 1 的資料，倒不如用“重複一千次的 1”來得省。這種壓縮技巧大幅使用在影像壓縮方面，因為在影像上，我們常常有相同的色彩在一片鄰近的位置上。

最近顛覆整個唱片娛樂業的 MP3，基本上是一種數位音樂的壓縮技術，它讓我們可以在一片光碟片上，儲存約十片 CD 唱片的音樂，也讓我們可以更便捷地從網路上下載喜愛的音樂，真是不同凡響。

WinZip 壓縮軟體是目前在個人電腦世界裡，最風行的一個資料壓縮軟體。我們現在從網路上下載的軟體，在使用前，幾乎都要先用 WinZip 來還原才行。它所使用的最主要壓縮格式是 zip 檔，而 zip 格式的發明人是卡茲，卡茲同時也是 WinZip 風行前，廣受歡迎的 PKZip 壓縮軟體的創作人。【2000/8/4 中國時報浮世繪版 數位世界說法 專欄-資料壓縮術 -- 親愛的，我把資料變小了 -- 趙坤茂】

資料壓縮系統

為達到資料壓縮的目的，通常我們必須找出存在原始資料間的相關特性，也就是存在原始資料間的資料累贅(data redundancy)。如果我們能找出這些累贅（或稱多餘資料），減少且移除這些累贅，就能達到資料壓縮的目的。

一個壓縮方法的編碼效率(coding rate)或稱壓縮效率(compression ratio or rate, Cr)通常使用下列方式來評估：

Cr= (原始資料的大小 － 壓縮後的資料大小 / 原始資料大小) × 100 %

而一般資料壓縮系統(data compression system)包含兩大部分：壓縮器(compressor)及解壓縮器(decompressor)。壓縮器主要的工作包含：(1)找出待壓縮資料（或稱做原始資料或資料原，source data or original data）中存在的資料累贅(data redundancy)。(2)移除這些累贅。(3)將剩餘的必要資料編碼並送出編碼結果（或稱壓縮結果，compressed data）等動作。相反的，解壓縮器的主要工作則是將讀入的編碼結果依據既定的步驟解碼並送出解碼結果（或稱為重建結果，decompressed data or reconstructed data）。

壓縮器主要功能在於壓縮編碼，故有時也稱為編碼器(encoder)，壓縮過程也可稱為編碼(encoding or coding)；同樣的，解壓縮器也稱做解碼器(decoder)，其過程亦可稱做解碼過程(decoding)過程。

無失真與失真壓縮

無失真壓縮(lossless compression)又稱做無損壓縮或是可逆壓縮(reversal compression)，由於重建結果和壓縮前的資料源完全相同，一般用於文字檔、執行檔、醫學影像和重要資料的壓縮上。為確保無失真，此類壓縮通常無法得到很好的壓縮率。設計一無失真的壓縮需考慮下列三因素：編碼效率（coding efficiency）、編碼延遲（coding delay）與編碼器複雜度（coder complexity）。編碼效率指的是壓縮效率，編碼延遲是指資料壓縮編碼所花費的時間，而編碼器複雜度則是指實現壓縮編碼演算法所需運算的複雜程度。當我們要求壓縮效率好一點時，通常編碼器複雜度與編碼延遲都會增加；相對的，如果希望編碼延遲端一點，通常效率會降低。如何在三者間找出一平衡點，而設計出一適合的要求的壓縮方法，是需要思考的。

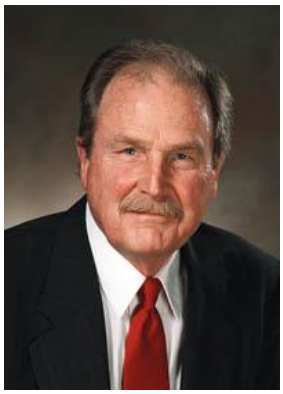
失真壓縮又稱為有損壓縮或是不可逆壓縮（irreversal compression），即重建結果和壓縮前的資料源不完全相同（也就說會發生資料短少的現象）。當然，重建結果雖和原始資料不同，但原始資料的大部分資訊與特色仍包含在重建的結果當中。由於失真壓縮的壓縮效率遠高於無失真壓縮，故常用於需要高壓縮比且允許部分失真的影像壓縮（如 JPEG 和 MPEG）與聲音壓縮(如 MP3) ，這部分後面將會有更為詳細的述說。當設計一個失真壓縮方法時，序要在下列四個因素中取得平衡：編碼效率、編碼延遲、編碼器複雜度與重建資料的品質(quality)。

資料壓縮過程

一般而言，資料是由許多連續的符號(symbols)所組成而成，通常這些符號間必存在著某種關連性，也就是說存在許多所謂的資料累贅（data redundancy）。如果我們能找出這些累贅，解少且移除這些累贅，就能達到資料壓縮的目的。

資料壓縮的過程，主要包含兩步驟：編碼(coding)及模式化(modeling)。所謂模式化是指萃取出(extract)資料中的累贅，並且選擇一個適當的模式來描述這些累贅的動作過程。而編碼則是將所選擇的模式之描述(model description)及資料與模式間的差異（或稱誤差），加以編碼（通常編成二進制碼）的動作過程。這兩個步驟的好壞，影響整個系統的壓縮效率，換句話說，選擇好的模式與好的編碼法可確保高效率的壓縮。因此，要設計一個實用且高效率的資料壓縮方法，必須妥善選擇所採用的編碼及模式。

霍夫曼編碼法（Huffman's Encode）



霍夫曼在1952年所提出的一種無失真壓縮技術，它的原理是將要壓縮之字串，先讀一遍，再將字串中的每一個相異單字元的出現頻率，做成統計，依此來建構霍夫曼樹。每一相異的字元，用 0 與 1 編碼，出現次數最多者，給較少的位元編碼，最後將這些位元串組合起來，並加上霍夫曼樹，就成為壓縮檔案。

霍夫曼編碼法是資訊源符號出現機率，在對資訊源符號逐一編碼條件下，最好的編碼方法。

霍夫曼編碼演算法：

- 步驟一:計算每一個符號出現的機率，然後把所有符號跟機率放入要處理符號集合 R 中，準備由(下往上)建立一棵編碼二元樹。
- 步驟二：從要處理的集合中找出機率最小的兩個符號做為二元樹的兩個子節點，並為這兩個節點建立一個父節點，此父節點機率為兩個子節點的機率和。再將這兩個子節點從 R 中移除，且把其父節點(含機率)加入 R 中。
- 步驟三：重複步驟二直到待處理集合只剩下一個符號。
- 步驟四：將建立完成的二元樹中任何兩兄弟節點的左邊線標上 0 右邊線標上 1。樹中每個符號被指定之 0 與 1 的位元集合即是代表該符號的字碼。
- 步驟五：使用在步驟四決定的符號字碼，一一編碼所有待壓縮符號。

霍夫曼編碼法在許多標準中被採用，最著名的就是應用於失真影像壓縮標準的 JPEG 中。在 JPEG 標準中，影像被切割為 8x8 的區塊，每一區塊各自進行轉換編碼，轉換後的 64 個係數就是使用霍夫曼編碼法加以編碼、壓縮。

霍夫曼樹:

常常被拿來處理大量的符號編寫工作。根據整組資料中符號出現的頻率高低，決定要給這個符號怎麼編碼。如果符號出現的頻率太高，則給符號的碼越短，相反符號的號碼越長。常常用來處理資料壓縮的問題。霍夫曼碼是由 0 和 1 所組成的，左子樹填入 0，右樹填入 1。

例如：

一組簡單的的英文單字：

K	N	G	I	H	E
4	2	5	1	3	7

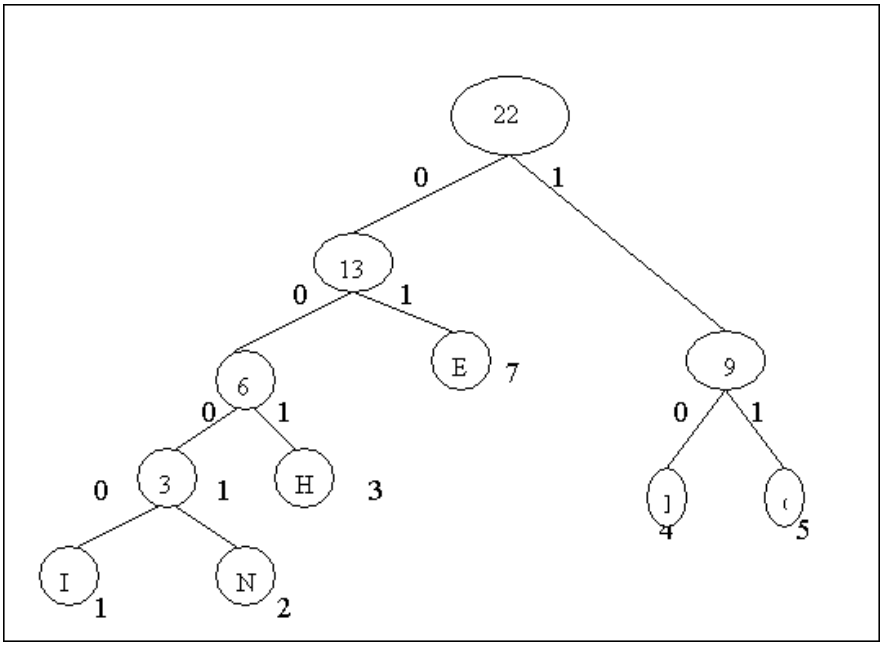
演算過程：

(一) 霍夫曼編碼是由小排到大，小的放左，大的放右。

由 K.N.G.I.H.E 相比，最小為 I(1)，次小為 N(2)，相加上去為 3。3 在與剩下的 K.G.H.E 相比，H(3)為最小，兩者放丨起，相加上去為 6，6 與剩下的 K.G..E 相比，K.G 比他小，卻比 H(3)大，因此

必須放在 H 的右邊，K.G 相加為 9，剩下 E(7)比 6 大，因此放 6 右邊，相加為 13，13 在與 9 相加上去，便是樹狀圖的完成了。

(二) 接下來在樹狀圖的左邊線標上 0，右邊線標上 1。然後照英文單字左至右，循邊線的 0 與 1 位元集合開始編碼。



K=10
N=0001
G=11.
I=0000
H=001
E=01

練習 1:假若我們有一組符號集 A={X, Y, Z}且我們為這組符號所編定之 Codeword 分別 為{0, 10, 11}。請問傳送端若依此 Codebook 送出一位元流「01001100」，則接收 端解碼後之符號為何（What's the decoded symbol sequence）？

提示：此一位元流（bit stream）解碼後應有六個符號！

Huffman Codes Example

- 假設有一段文字由 a,b,…g 這些字母組成，其出現頻率分別如下：

characters	Freqency	ASCII(8bits)
a	37	
b	18	
c	29	
d	13	
e	30	
f	17	
g	6	

Q1:若以 ASCII(8bits) 進行編碼，所需位元數共多少?

A1:

- 試建構一 Huffman tree(過程寫於背面)，重新將 a,b…g 進行編碼，並填入下表

characters	Freqency	Huffman codeword
a	37	
b	18	
c	29	
d	13	
e	30	
f	17	
g	6	

Q2:若以 Huffman code 進行編碼，所需位元數共多少?

A2:

Q3: 試計算壓縮率

A3:

Huffman tree

Huffman Tree – Python code

```
import queue

class HuffmanNode(object):
    def __init__(self,left=None,right=None,root=None):
        self.left = left
        self.right = right
        self.root = root
    def children(self):
        return (self.left,self.right)
    def preorder(self,path=None):
        if path is None:
            path = []
        if self.left is not None:
            if isinstance(self.left[1], HuffmanNode):
                self.left[1].preorder(path+[0])
            else:
                print(self.left,path+[0])
        if self.right is not None:
            if isinstance(self.right[1], HuffmanNode):
                self.right[1].preorder(path+[1])
            else:
                print(self.right,path+[1])

freq = [
    (8.167, 'a'), (1.492, 'b'), (2.782, 'c'), (4.253, 'd'),
    (12.702, 'e'),(2.228, 'f'), (2.015, 'g'), (6.094, 'h'),
    (6.966, 'i'), (0.153, 'j'), (0.747, 'k'), (4.025, 'l'),
    (2.406, 'm'), (6.749, 'n'), (7.507, 'o'), (1.929, 'p'),
    (0.095, 'q'), (5.987, 'r'), (6.327, 's'), (9.056, 't'),
    (2.758, 'u'), (1.037, 'v'), (2.365, 'w'), (0.150, 'x'),
    (1.974, 'y'), (0.074, 'z') ]

def encode(frequencies):
    p = queue.PriorityQueue()
    for item in frequencies:
        p.put(item)
    #invariant that order is ascending in the priority queue
    #p.size() gives list of elements
    while p.qsize() > 1:
        left,right = p.get(),p.get()
        node = HuffmanNode(left,right)
        p.put((left[0]+right[0],node))
    return p.get()

node = encode(freq)
print(node[1].preorder())
```

Hint:

Writing our first file

```
myfile = open("test.txt", "w")
myfile.write("My first file written from Python\n")
myfile.write("--------------------------------\n")
myfile.write("Hello, world!\n")
myfile.close()
```

Reading a file line-at-a-time

```
mynewhandle = open("test.txt", "r")
while True:                          # Keep reading forever
    theline = mynewhandle.readline()   # Try to read next line
    if len(theline) == 0:             # If there are no more lines
        break                        #     leave the loop

    # Now process the line we've just read
    print(theline, end="")

mynewhandle.close()
```

Reading the whole file at once

```
f = open("test.txt")
content = f.read()
f.close()

words = content.split()
print("There are {0} words in the file.".format(len(words)))
```

File Method

```
FileObject = open(filepath,mode)
f = open("test.txt", "w")
f.close()
f.read()
f.write()
f.readline()
```

YOUR TASK：計算文字檔案中每個字元出現頻率

```
mynewhandle = open("test.txt", "r")
count = 0
freq=dict()
while True:                          # Keep reading forever
    theline = mynewhandle.readline()   # Try to read next line
    count += len(theline)
    for c in theline:
        if c in freq:
            freq.update({c:freq[c]+1})
        else:
            freq.update({c:1})
    if len(theline) == 0:            # If there are no more lines
        break                       #    leave the loop
    # Now process the line we've just read
    print(theline, end="")
print(count)
print(freq)

for key in freq.keys():
    print(freq[key]/count*100,end='\t')

mynewhandle.close()
```

字典(DICTIONARY)：dict (https://openhome.cc/Gossip/Python/DictionaryType.html)

在 Python 中，字典物件 dict()是儲存鍵（Key）/值（Value）對應的物件。與 list 差異如下：

List

index	value
0	"Eggs"
1	"Milk"
2	"Cheese"
3	"Yogurt"
4	"Butter"
5	"More Cheese"

Dictionary

key	value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter'	2.59
'More Cheese'	6.19

直接示範如何以實字建立字典物件：

```
>>> passwords = {'Justin' : 123456, 'caterpillar' : 933933}
>>> passwords['Justin']
123456
>>> passwords['caterpillar']
933933
>>> passwords['Hamimi'] = 970221
>>> passwords
{'caterpillar': 933933, 'Hamimi': 970221, 'Justin': 123456}
>>> passwords['Hamimi']
970221
>>> del passwords['caterpillar']
>>> passwords
{'Hamimi': 970221, 'Justin': 123456}
```

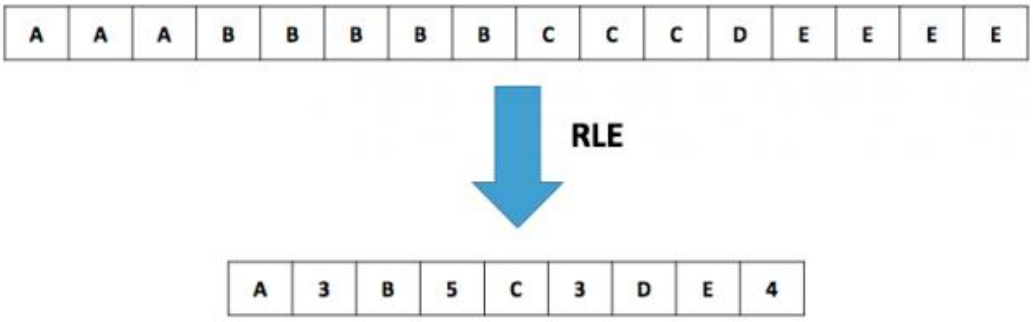
update()將指定的字典物件加入，使用 **pop()**方法可以指定鍵將對應的鍵/值取出並從字典中移除

Python Dictionary Methods

Method	Description
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v(defaults to None).
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

Computational Thinking & Programming
資料壓縮技術 Data Compression

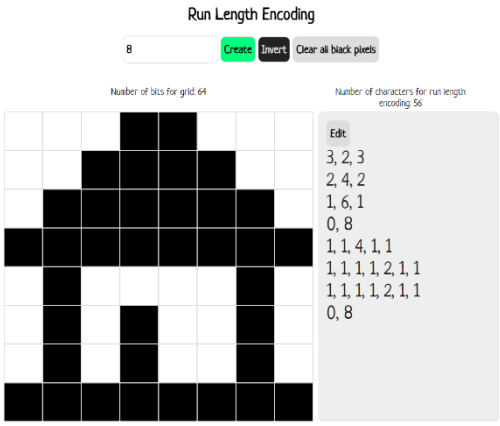
Concepts: Data presentation, data compression

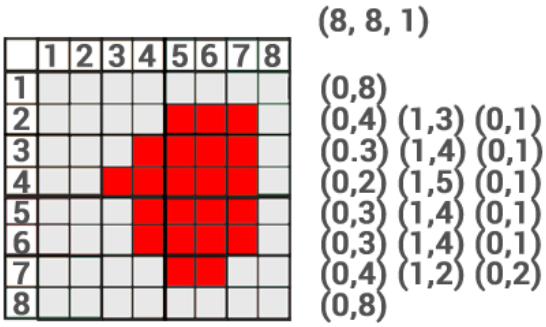


變動長度編碼法（RLE, Run Length Encoding）

The one simplest compression algorithm is called Run-Length. It is just counting the number of identical sequential characters and note down each time as a pair of appearance and its counter. For example, if a given text is *aaabbccc,* it can be shorten to *a3b2c3.* This is great if you have texts such as *aaaaaaaaaa … (one more million times)* . However, on the contrast, if a very simple text is like this *abcde,* the compression algorithm will not do any good. Rather it wastes another byte to store the number one. This is why best compression software such as WinRAR implements several combination of compression algorithms, not just a single one.

The idea of the Runlength compression algorithm can be illustrated by the following Python code, simple and elegant. The complexity of the algorithm is efficient with simple pass $O(n)$.





Runlength Compression Algorithm, Demonstration in Python

```
def runlen(s):
    r = ""
    l = len(s)
    if l == 0:
        return ""
    if l == 1:
        return s + "1"
    last = s[0]
    cnt = 1
    i = 1
    while i < l:
        if s[i] == s[i - 1]: # check it is the same letter
            cnt += 1
        else:
            r = r + s[i - 1] + str(cnt) # if not, store the previous data
            cnt = 1
        i += 1
    r = r + s[i - 1] + str(cnt)
    return r


def main():
    while True:
        ss = input();
        if ss==="":
            break
        print(runlen(ss))


main()
```

Your Task: Decoding Run Length Code