

HW2

一、(10 分)

1. 最長的 argument 至少包含 3 種運算元 4 種運算子且推論正確，需逐步說明(或真值表) (5 分)

錯誤與不完整斟酌扣分

2. 帶入實例說明(2 分)

3. 複雜度，解題方式與討論的廣度，如果一開始的 argument 就寫比較簡單，這項分數就會比較低(3 分)

二、(10 分)

本題是針對程式執行上的困難點大致可分成以下幾種，

a. 語意問題(切割, 順序, 自然語言...)

b. 單一路徑問題(複雜度, 優化, rule 選擇...)

c. 無法推論出來的情況

部份給分，完整的寫出一項可以得到 5 分，完整寫出兩項完全不同的困難點 (ex 切割+路徑複雜度) 可得 10 分。相近的困難點 (ex 複雜度+優化) 分項得分會比較少。

HW3

一、

1. 寫出 Gray Code 規則(相差 1bit)、遞迴產生、該例只是其中一種 Gray Code、集合觀點

2. 寫出除了 Example 外的應用、延伸(像遞迴產生 Gray Code 文章就有寫了，不算分)，並描述如何應用、延伸

3. 越豐富越好，有舉例更好

二、

1. 提出新的規則，並且多於 4 個、不要只有一種組合

2. 寫出如何實作 或 詳細描述困難的點在哪，以數量級表示、估算為佳

- P: 玩電腦
 Q: 寫作業
 R: 想睡覺
 S: 聽音樂
 T: 運動
 U: 精神好

5
2
3

$$[(\neg P \vee Q) \rightarrow R] \wedge [R \rightarrow (S \vee T)] \wedge [\neg U \rightarrow \neg T] \wedge [\neg S \wedge \neg U] \rightarrow P$$

- $(\neg P \vee Q) \rightarrow R$ 我不玩電腦或寫作業，就會想睡
 $R \rightarrow (S \vee T)$ 我想睡覺，就會聽音樂或運動
 $\neg U \rightarrow \neg T$ 我精神不好，就不會運動
 $\neg S \wedge \neg U$ 我不聽音樂且精神不好
 $\therefore P$ 因此，我在玩電腦

過程：(1) $\neg S \wedge \neg U$ 前提

(2) $\neg U$

(1) Rule of Conjunctive Simplification

(3) $\neg U \rightarrow \neg T$ 前提

(4) $\neg T$

(2)(3) Rule of Detachment

(5) $(\neg P \vee Q) \rightarrow R$ 前提

(6) $R \rightarrow (S \vee T)$ 前提

(7) $(\neg P \vee Q) \rightarrow (S \vee T)$ (5)(6) Law of Syllogism

(8) $(\neg S \wedge \neg T) \rightarrow (P \wedge \neg Q) \Leftrightarrow (7)$

(9) $\neg S$

(1) Rule of Conjunctive Simplification

(10) $\neg S \wedge \neg T$

(4)(9) Rule of Conjunction

(11) $P \wedge \neg Q$

(8)(10) Rule of Detachment

(12) P

(11) Rule of Conjunctive Simplification

2. 非常困難，有以下的問題

- (1) 有多個不同的敘述：程式須接收不定數量的敘述，且每個敘述可以很簡單也可以很複雜地由多種敘述而組成
- (2) 答案並不唯一：可以得到多種不同的推論結果，得出的結果不一定是我們關注的
- (3) 多種不同的解法：相同的推論結果，可能由不同的方法來推出，由於有多種規則可以使用，很難制定一個通用的方法決定規則間配合敘述使用的先後順序。

#1. Write an argument with proof & description.

10 △ Argument: $((p \vee q) \vee r) \wedge \neg(q \vee r) \rightarrow p$

where $a \vee b := (a \vee b) \wedge \neg(a \wedge b)$. (as the lecture slide)

△ Proof: Denote T=1, F=0, and let LHS of the argument be ①.

$p \vee q \vee r$	$(p \vee q) \vee r$	$\neg(q \vee r)$	①	$\neg(\neg(p \rightarrow q))$
0 0 0	0	1	0	1
0 0 1	0	1	0	1
0 1 0	1	1	0	1
0 1 1	1	0	0	1
1 0 0	1	1	1	1
1 0 1	1	0	0	1
1 1 0	0	0	0	1
1 1 1	0	1	0	1

5
2
3

Hence, for each possible 3-tuple $(p, q, r) \in \{T, F\}^3$,
 $\neg(\neg(p \rightarrow q))$ holds.

△ Description: Apple 和 Samsung 在扭打，老師可能會來制止。

p : Apple 打贏, $\neg p$: Apple 沒打贏

q : Samsung 打贏, $\neg q$: Samsung 沒打贏

$p \vee q$: Apple & Samsung 恰有一人打贏

r : 在分出勝負前就被老師制止

「要嘛就恰有一人打贏，要嘛就被老師喝止」

但又不是「Samsung 打贏或老師喝止」，故 Apple 勝。

△ Note

1. 我不是果粉，名稱雷同純屬巧合。
2. 証明可用 AND 的 associative law 簡化，但這件事在
(XOR的結合律)

命題邏輯中並沒有這麼明顯。

如果將真假值與 Exclusive OR 視為循環群 \mathbb{Z}_2 ，則

結合律是必然的，證明可參考：

$$\text{prove } ((p \vee q) \vee r) \wedge \neg(q \vee r) \rightarrow p:$$

Lemma 1: $(a \vee b) \text{ implies } a \vee b$

Lemma 2: $(a \vee b) \vee c \Leftrightarrow a \vee (b \vee c)$, (associative).

$$\text{proof: } p \vee (q \vee r) \wedge \neg(q \vee r) \rightarrow p \quad \text{為 T.}$$

By Lemma 2,

$$(p \vee q) \vee r \wedge \neg(q \vee r) \rightarrow p \quad \text{為 T.}$$

By Lemma 1,

$$(p \vee q) \vee r \wedge \neg(q \vee r) \rightarrow p \quad \text{為 T.}$$

#2. How to write a program to make inference automatically.

10

△ 技術面：^①對於所有變數都已賦值的邏輯式，可以使用後序演算法搭配 stack 來算出其真值，如

$$(((T_0 \vee F_0) \vee T_0) \wedge (T_0 \wedge F_0) \rightarrow F_0) \wedge F_0$$

可得到 $false(F_0)$ 。

^②但對於推導事實的部分，則有所不同。首先可用上述方法搭配暴力法，如欲知

$$(p \vee q \vee r) \wedge \neg(q \vee r) \rightarrow ?$$

可推出什麼結果，則將 p, q, r 代入各種真值，
套入 \oplus ，並收集結果為 T 的配對，得到

p 必發生而 q, r 必不發生。

但 ^③的 time complexity 為 $\Omega(2^N)$ ， $N =$ 自變數的量。
事實上 satisfaction problem 是 NP-Complete 的，同

時也是電路簡化的關鍵問題。

③ 對 User input 的預處理方面，pushdown automata 可

做到文法檢查，利用現有的簡化公式也可加快後續執行速度。

④ 關於謂詞邏輯上的推論應該也是可能的，但更複雜。

△ 現有工具

① 使用 prolog 的 backward reasoning，則命題邏輯中不太困難的 inferencing 應該能辦到

② 若要實現 [技術面①] 中的功能，可直接使用 javascript 中的 eval(...) 或類似工具。

#1. About Ex 3.9

△ What does this example say?

這個例子給出了系統性構造格雷碼的方法，且是遞迴式的。如果欲將 $\{0, 1, \dots, 2^n - 1\}$ 編為 Gray code 且 $\{0, 1, \dots, 2^{n-1} - 1\}$ 已編碼完成，則 $\{0, 1, \dots, 2^{n-1} - 1\}$ 的部分要在前面補零， $\{2^{n-1}, \dots, 2^n - 1\}$ 的部分則為 $0 \sim 2^{n-1} - 1$ 的反序，且把 MSB 改為 1。

同時課本也把 Gray code: $\{0, 1\}^n$ 映射到 cardinality 為 n 的集合 S 的幕集合 $\mathcal{P}(S)$ 。由於是 bijection，因此說 $|\mathcal{P}(S)| = 2^{|S|}$ 。這裡可能有借用到 indicator function (characteristic function) 的概念：所有 $T \subseteq S$ 都可視為一個函數 $F_T: S \rightarrow \{0, 1\}$ ，其中 $F_T(t) = \begin{cases} 1 & \text{if } t \in T \\ 0 & \text{if } t \notin T \end{cases}$ 。（因此對 $p \in \mathcal{P}(S)$ 編碼不一定要用 Gray code）

最後它也介紹其它種可能的 Gray code。

△ Extension & Application?

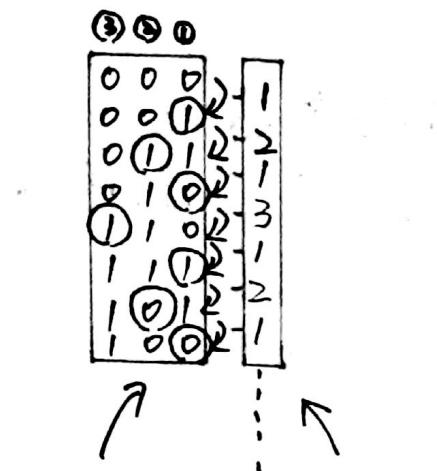
擴充：我們可以讓位元為 0, 1 以外的數，亦即在 $\{0, 1, \dots, k^n - 1\}$ 及 $\{0, 1, \dots, k-1\}^n$ 之間構造一個 bijection f ，使得 $\forall m < k^n - 1$ ， $f(m+1)$ 與 $f(m)$ 只差一個 digit，稱為 n -ary gray code。

應用：^① 傳送訊號有一個 bit 傳輸錯誤 (single bit flipping) 時，比一般的編碼有更大的機會被解讀成差異不大的數字。

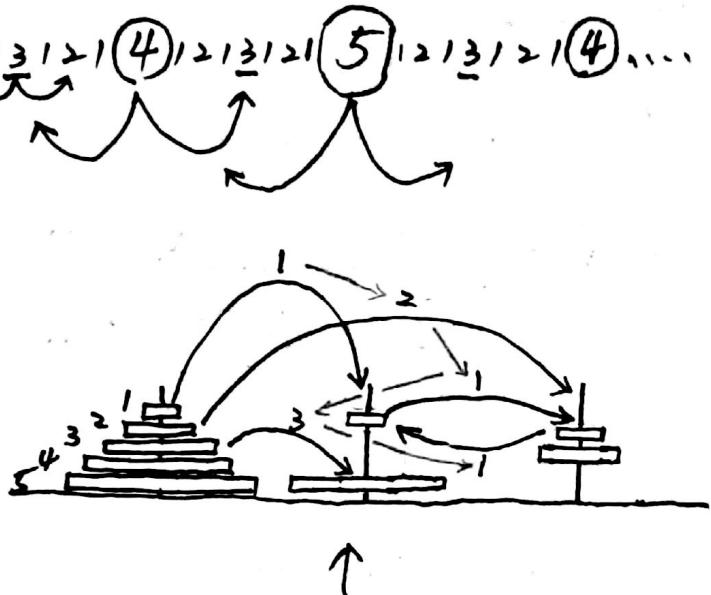
② 令電路更快速的進行 +1, -1. 的運算，因為只
需更動一個位元。

③ 依序記錄 gray code 改動的位元位置，可以得到

Gros Sequence: 1213121(4)1213121(5)1213121(4)...



Gray Code Gros Seq

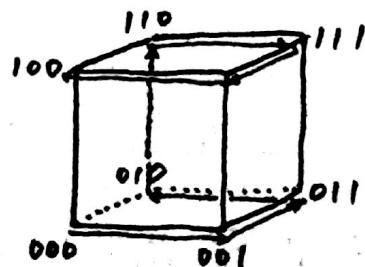


Hanoi Tower

而 Gros Sequence 正是 Tower of Hanoi 每步要移動的圓盤編號。(這裡的 Gray Code 是課本方法所定出來的才行)。

此外 Gros Sequence 尚有其良好性質, eg, greedy non-repetitive sequence.

④ 格雷碼也給出 n -dimensional unit cube 上的 Hamiltonian path (or cycle):



← 三維立方體上 Hamiltonian Path
的例子。

△ What do we get after this reading?

雖然格雷碼在上就敘過了，不過作者特地把格雷碼對應到幕集合 $\mathcal{P}(S)$ 的用意難以參透。如果只是要解釋 $|\mathcal{P}(S)| = 2^{|S|}$ ，並不需要用到格雷碼，任何編碼都可以。

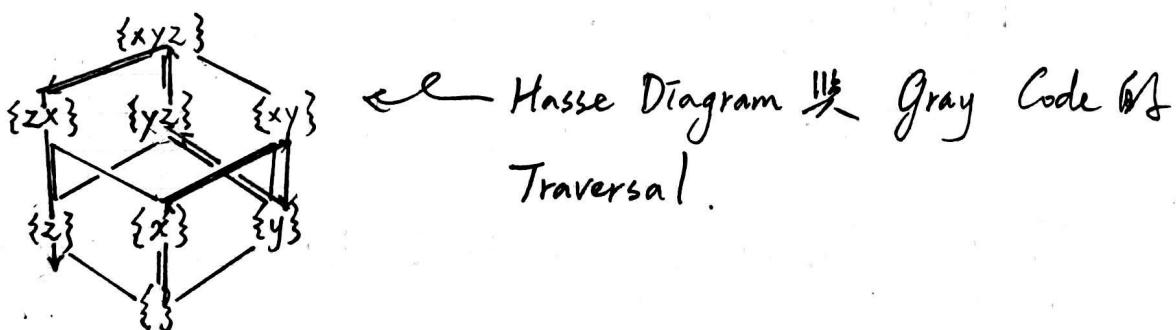
有據於此，我們可以思考：用格雷碼來為 Subset 編碼，究竟多了什麼？

顯然它是對 $\mathcal{P}(S)$ 的一個 traversal！

$$\mathbb{Z}_{0 \leq i < 2^3} \xrightarrow{\text{gray}} \{0, 1\}^3 \longrightarrow \mathcal{P}(S)$$

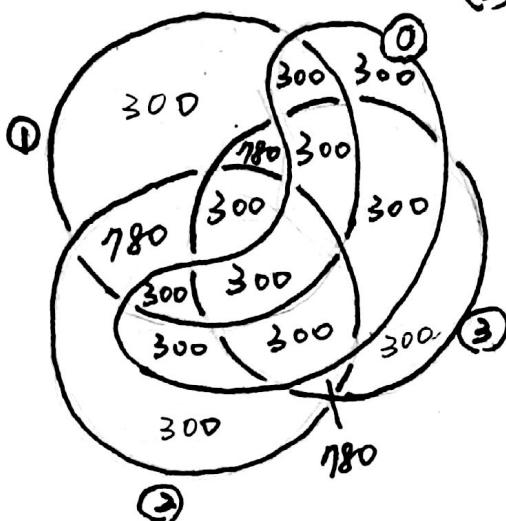
where $S = \{x, y, z\}$

且上一步與下一步之間只差一個元素，顯然兩步之間有包含或被包含的關係，故它們在 Lattice : $(\mathcal{P}(S), \subseteq)$ 之下相鄰，亦即在 Hasse diagram 下相鄰。我們可以畫出 traversal 的過程



雖然是有向圖，但如果無視方向，則和三維立方体有同構。

#2. "T,O,P,O,L,O,G,T". for 'O's not adjacent, or 'T' before 'L',
or 'L' before 'G', or 'G' before 'T'.



$$\textcircled{1} = \textcircled{2} = \textcircled{3} = \frac{8!}{3!2!} = 3360, \quad \textcircled{0} = 5! \times C_3^6 = 2400$$

$$\textcircled{1} \cap \textcircled{2} = \frac{8!}{3!2!2!} = 1680 = \textcircled{2} \cap \textcircled{3} = \textcircled{3} \cap \textcircled{1}$$

$$\textcircled{1} \cap \textcircled{0} = \frac{5!}{2!} \times C_3^6 = 1200 = \textcircled{2} \cap \textcircled{0} = \textcircled{3} \cap \textcircled{0}$$

$$\textcircled{1} \cap \textcircled{2} \cap \textcircled{0} = \frac{5!}{2!2!2!} \times C_3^6 = 600 = \textcircled{2} \cap \textcircled{3} \cap \textcircled{0} = \textcircled{3} \cap \textcircled{1} \cap \textcircled{0}$$

$$\textcircled{1} \cap \textcircled{2} \cap \textcircled{3} \cap \textcircled{0} = \frac{5!}{2!2!2!2!} \times C_3^6 = 300.$$

Fill the diagram \Rightarrow Sum = $3 \times 780 + 12 \times 300 = 5940$.

△ About programming :

如以上寫法，我們需取得任 k 個集合交集的 cardinality, ($k \in \{1, 2, \dots, n\}$)。如果這些交集不如下面的例子對稱，則至少需要 $\sum_k C_K^n \sim \Omega(2^n)$ 的 time & space.

推出每個小塊區域 Cardinality 至少要大於小塊數量的時間，故也至少 $\Omega(2^n)$ 。

gray [6+2] code

	Code 2 → 3	or
1 →	0 0 1 0	
2 →	1 0 1 0	
3 →	1 1 1 0	
4 →	0 1 1 0	
5 →	0 1 1 1	
6 →	1 1 1 1	
7 →	1 0 1 1	
8 →	0 0 1 1	

(a)

0 0 0	0 1 0	0 1 1
0 0 1	1 0 1	1 1 1
1 0 1	1 1 0	1 1 0
1 0 0	1 0 0	

(b)

* 將 A[1]-1 處 code 到下一個 code, 每次都只改變 1 bit 的值, 而且在遞迴的特性可從長度 2 改變為長度 3 的 gray code (如左圖 a)

* → 若以 gray code 儲存資訊, 當使用 Sensor 來讀取資訊時, 因為一次只能改變 1 bit 則當 sensor 因為不夠敏感或精確, 使得讀取資訊出錯時, gray code 可以偵錯 (會發生 2 bits 改變)

→ gray code 在跨時域的交界區域, 可以編碼出異步 FIFO 的 pointers (first-in-first-out)

→ gray code 也可使用在通信, 可最小化噪音對串位通訊的影響.

* gray code 受歡迎的原因不外乎其每次只改變 1 bit 的特色, 用來偵錯或加速搜尋都十分方便, 合乎現代科技所追求的 2 個重點 - more fast, less error. 而 gray code 的缺點在於人腦難以快速判斷 (慣性思考), 但對於電腦, gray code 的缺點則不存在, 反而更加快速。

< How do you write a code .. >

19 → If n sets, 若有某些條件有相關 (如 n 在 d 之前, d 在 s 之前 ...), 則在判斷 A&B 或 B&C ... 的情況會使排列數不相同,

又若有字母相同情況 (如對 HOMEWORK 作排列), 則要考慮相同的 "O", 在判斷關於 O 的條件有更多限制.

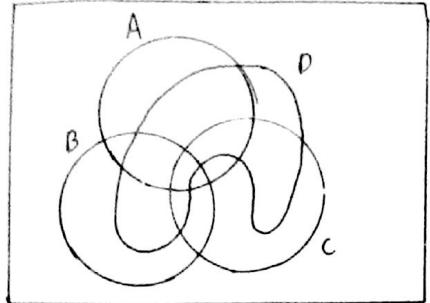
又若 n sets 不是相似 (如 n 在 d 之前, d 和 s 相鄰), 則要考慮 A&B 或 B&C ... 的情況排列數不相同.

, 售有在 n 個條件都相同, 且不相關則可有公式.

Ex. 若 n sets, 且都為 A 在 B 之前, C 在 D 之前 ..., 有 m 個 letters

$$\text{排列} \rightarrow C_1^n \cdot H_{m-2}^3 \cdot (m-2)! - C_2^n \cdot H_{m-3}^4 \cdot (m-3)! + C_3^n \cdot H_{m-4}^5 \cdot (m-4)! - \dots + C_n^n \cdot H_{m-n-1}^{n+2} \cdot (m-n-1)!$$

2.



2.

(Venn diagram).

introduce 1° i 在 h 之前或 h 在 t 之前或 t 在 r 之前或 r 在 o 之前

A ↗

B ↗

C ↗

D ↗

$$\begin{aligned}
 \text{所求 } |A \cup B \cup C \cup D| &= |A| + |B| + |C| + |D| - |A \cap B| - |B \cap C| - |C \cap D| - |A \cap C| - |A \cap D| \\
 &\quad - |B \cap D| + |A \cap B \cap C| + |A \cap B \cap D| + |B \cap C \cap D| - |A \cap B \cap C \cap D| \\
 &= C_1^4 \cdot H_7^3 \cdot 7! - 3 \cdot H_6^4 \cdot 6! - 3 \cdot 2 \cdot H_5^5 \cdot 5! + 2 \cdot H_5^5 \cdot 5! + 2 \cdot 2 H_4^6 \cdot 4! \\
 &\quad \text{intro} \\
 &= 49229_{12} \quad (\text{計算和結果} \times b) \quad - H_4^6 \cdot 4!
 \end{aligned}$$

2° i 在 h 之前或 h 在 t 之前或 t 在 r 之前或 r 在 o 之前

A ↗

B ↗

C ↗

D ↗

$$\text{所求 } |A \cup B \cup C \cup D| = |A| + |B| + |C| + |D| - |A \cap B| - |B \cap C| - |C \cap D| - |A \cap C| - |A \cap D| - |B \cap D|$$

$$+ |A \cap B \cap C| + |A \cap B \cap D| + |B \cap C \cap D| - |A \cap B \cap C \cap D|$$

$$\begin{aligned}
 &= C_1^4 \cdot H_7^3 \cdot 7! - 2 \cdot H_6^4 \cdot 6! - 4 \cdot 2 \cdot H_5^5 \cdot 5! + H_5^5 \cdot 5! + 2 \cdot 2 \cdot H_4^6 \cdot 4! \\
 &\quad \text{int,od/int,od/int,od/tr,od} \quad \text{intr} \quad \text{int,od/int,od} \\
 &\quad + 3! \cdot H_3^7 \cdot 3! - H_3^7 \cdot 3!
 \end{aligned}$$

$$= 513576$$

3° Introduces

i 在 h 之前或 t 在 r 之前或 o 在 d 之前或 u 在 c 之前或 e 在 s 之前

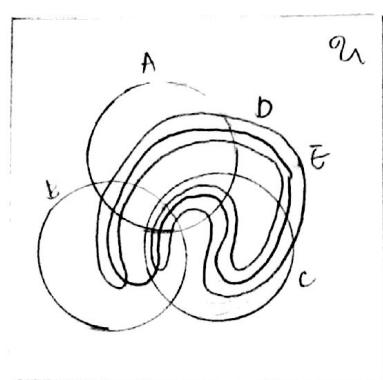
A ↗

B ↗

C ↗

D ↗

E ↗



venn diagram

$$\text{所求 } |A \cup B \cup C \cup D \cup E| = |A| + |B| + |C| + |D| + |E| - |A \cap B \cap C \cap D \cap E|$$

$$= C_5^5 \cdot H_8^3 \cdot 8! - C_2^5 \cdot H_7^4 \cdot 7! + C_3^5 \cdot H_6^5 \cdot 6! - C_4^5 \cdot H_5^6 \cdot 5!$$

$$+ C_5^5 \cdot H_4^7 \cdot 4!$$

$$= 3785040 *$$

< How do you write a code to calculate in 前-真 >.

1.10

(1) 根據來說教如何產生 Gray code，詳述方法如下

(a) 先產生一組且

0

1

(b) 將(a)的內容上下鏡射，並再上半部後方補一個0，下半部補1

鏡 射	0	0	0	補
	1	0	0	0
	1	1	0	補
	0	1	0	1

(c) 重複(b)的作法，先鏡射後上面補0，下面補1

鏡 射	0	0	0	補
	1	0	0	0
	1	1	0	補
	0	1	0	1
	0	1	1	補
	1	1	1	1

於是依照以上的方法，我們就可以用 recursive, 遞迴的方式不斷得到 size 更長的 Gray code。

至於什麼是 Gray code 呢？就是上下相鄰的兩組數字中，只會有一碼會是不同的。且依照以上規則得到的一整組 Gray code 並不會有重複的狀況產生，我們可以將 0 想成是 \emptyset ，而將 1 想成雙數，

(a)	0	\emptyset	(b)	0	0	{ \emptyset }	(c)	類推 x, y, z ...
	1	{x}		1	0	{x}		
				1	1	{x, y}		
				0	1	{y}		

可以發現因為它的不重複，也就是它會包含所有的子集（即(a)中就是 {x} 的所有子集，(b) 則是 {x, y} 的所有子集，(c) 則是包含 {x, y, z} 的...）

1. (2)

Gray code 最一開始的目的像是為了避免訊號傳遞錯誤而發明的，因為相隔的兩組數字間又會有一位元數字不同，是故當有訊號傳遞錯誤時就很明顯。因此多數用於二進制進 data 傳輸時的轉換。此外，一次只換一個位也在傳輸時較有效率。而 Gray code 的大小延伸也很容易，只要依照 (1) 中提到的方法就能輕易地擴大再擴大。

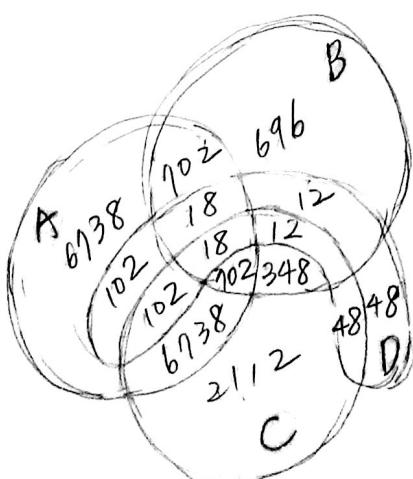
(3) 主要是學會了 Gray code 產生的一種容易的方式，此外，它更提供了找一個尋找所有 subset 的好方式！常常當 set 中的 element 一多時，就很容易在找 subset 時有漏找或是重複的現象，但現在搭酉灰 Gray code 的產生方式，就能輕易找出所有 subset 了！

2.

對於 "discrete" 這幾個英文字母的排序方式：

20

- 兩個 e 不相鄰
- "dis" 三個字不用依序但彼此相鄰
- c 排在 r 的前面 (不用相鄰)
- 出現 "cst" 這三個字相鄰且依序的狀況



- $a \wedge b \wedge c \wedge d = 3! \times C_2^4 = 18$
- $a \wedge b \wedge c \wedge \neg d = 3! \times 2! \times H_2^3 \times C_2^5 - \emptyset = 102$
- $a \wedge b \wedge \neg c \wedge d = 3! \times C_2^4 - \emptyset = 18$
- $a \wedge \neg b \wedge c \wedge d = 2! \times H_2^3 \times C_2^5 - \emptyset = 102$
- $\neg a \wedge b \wedge c \wedge d = \frac{3!}{2!} \times H_2^4 - \emptyset = 12$
- $a \wedge b \wedge \neg c \wedge \neg d = 3! \times 4! \times C_2^5 - \emptyset - \textcircled{3} = 102$
- $a \wedge \neg b \wedge c \wedge \neg d = 4! \times H_2^3 \times C_2^2 - \emptyset - \textcircled{2} - \textcircled{4} = 6738$
- $a \wedge \neg b \wedge \neg c \wedge d = 4! \times C_2^5 - \emptyset - \textcircled{3} - \textcircled{4} = 102$
- $\neg a \wedge b \wedge c \wedge \neg d = 3! \times \frac{4!}{2!} \times H_2^5 - \emptyset - \textcircled{2} - \textcircled{5} = 348$
- $\neg a \wedge \neg b \wedge c \wedge \neg d = \frac{4!}{2!} \times H_2^5 - \emptyset - \textcircled{4} - \textcircled{5} = 48$
- $\neg a \wedge b \wedge \neg c \wedge d = \frac{3!}{2!} - \emptyset - \textcircled{3} - \textcircled{5} = 12$
- $a \wedge \neg b \wedge \neg c \wedge \neg d = 6738$
- $\neg a \wedge b \wedge \neg c \wedge \neg d = 69b$
- $\neg a \wedge \neg b \wedge c \wedge \neg d = 2112$
- $\neg a \wedge \neg b \wedge \neg c \wedge d = 48$

3. (1) 假設我的程式可以有能力計算出各 set 的狀況：

若是在該情況下，只是要計算出各集合間的交集與差集個數，便可以畫出文氏圖的各區域，可以用排容原理來做到，理論上不會太困難。

(2) 假設我的程式連能否計算出某單一 set 的數量」這點都要考慮的話，則結果（複雜程度）就很難說了。畢竟在該種狀況下，困難的是能否計算 set 中的總 element 數量，倘若題目不難，則畫出文氏圖將不會有太大的困難，反之則否。

譬如(1)所述，(2)能否繪製文氏圖的關鍵為能否計算各 set 數，這乃是影響文氏圖是否能被畫出之關鍵。