

**1.**

*log(a,b) = log(a) + log(b) p(X=x) = ∑x p(X=x,Y=y) p(a,b) = p(a|b)\*p(b)*

*H(X,Y) = -∑x∑y (p(x,y)\*log2(p(x,y)))*

*= -∑x∑y (p(x,y)\*log2(p(x|y)\*p(y)))*

*= (-∑x∑y (p(x,y)\*log2(p(x|y)))) + (-∑x∑y(p(x,y)\*log2(p(y))))*

*= (-∑x∑y (p(x|y)\*p(y)\*log2(p(x|y)))) + (-∑x∑y(p(x,y)\*log2(p(y))))*

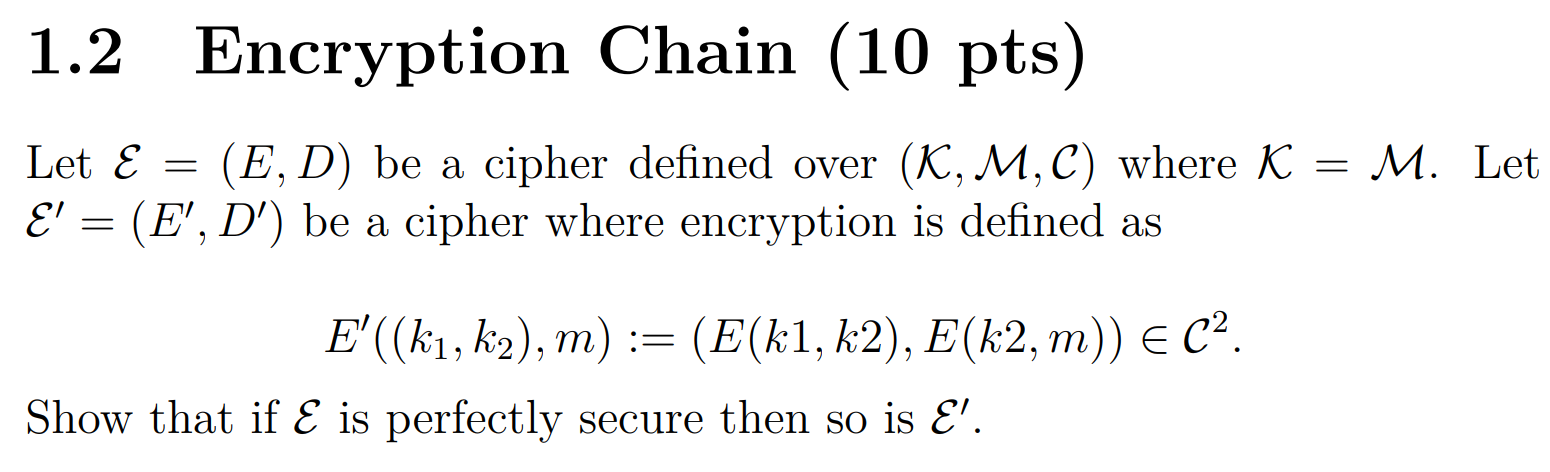
*= (-∑y(p(y))∑x(p(x|y)\*log2(p(x|y)))) + (-∑y(p(y)\*log2(p(y))))*

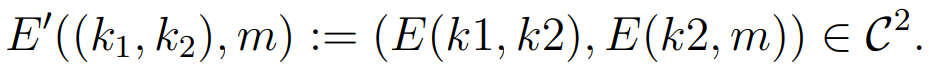
*= H(X|Y) + H(Y)*

**2.**

結合1.可知 *H(Y) + H(X|Y) ≤ H(X) + H(Y)*，故變成證明*H(X|Y) ≤ H(X)*

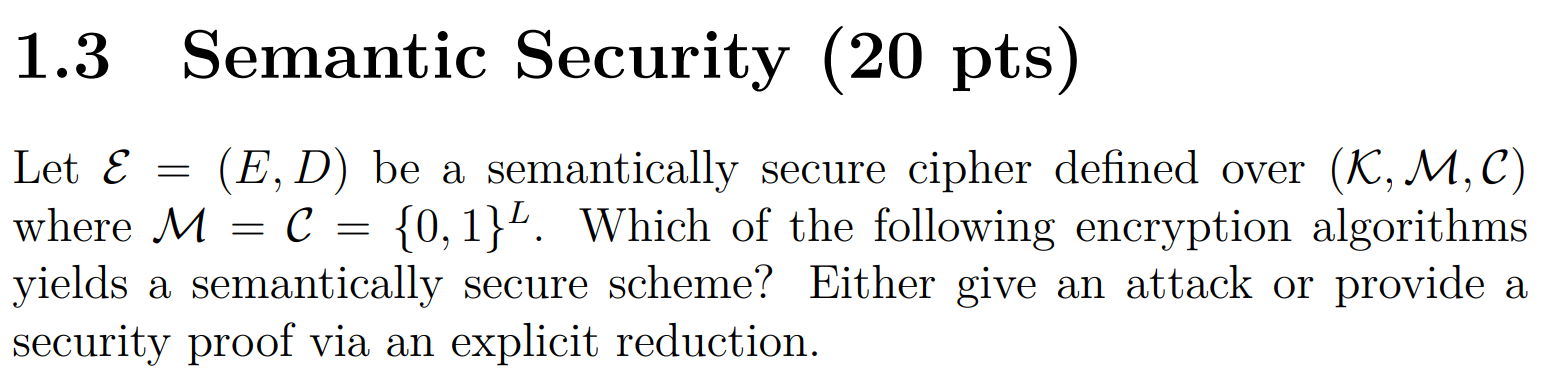
因為H(X|Y)表示「給了Y的資訊後, X剩下的資訊量」，所以他必少於或等於「X本身的資訊量」H(X)

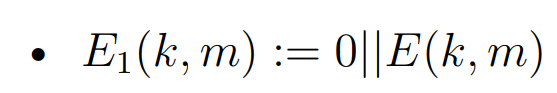


 這句的意思是先用k1對m加密(c1)，再將結果用k2加密(c2)

*m -> c1 -> c2*

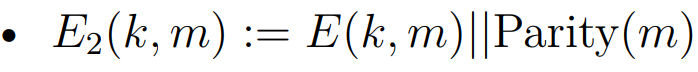
ε是完美安全，代表觀察c1無法得知m的資訊，且觀察c2也無法得知c1的資訊，代表觀察c2無法得知m的資訊，故也是完美安全



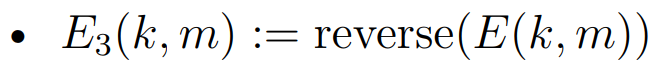
 在加密結果前面加一個0

這**是語意安全**，因為在結果前面加0不會對*E(k,m)*的進行有影響，而且無論如何都加0，對遊戲中的攻擊者分辨沒有幫助

*P[Ek(m0) = c] ≈ P[Ek(m1) = c]*

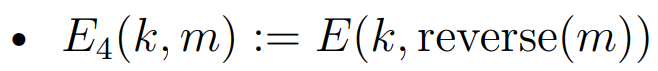
 把加密的結果和m的奇偶性合起來

這**不是語意安全**，因為遊戲的時候，只要攻擊者讓m0和m1的奇偶不同，就可以100%判斷是哪個

 把加密的結果反轉

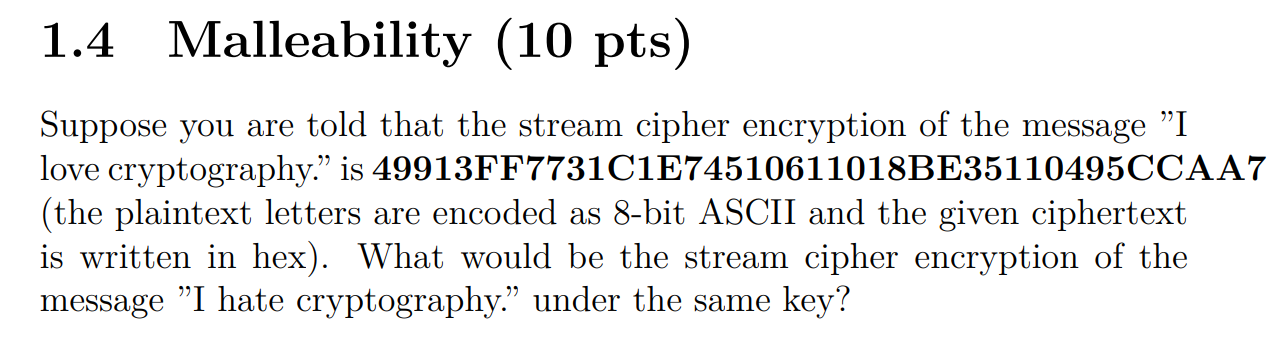
這**是語意安全**，因為反轉加密結果對*E(k,m)*沒有影響，遊戲中攻擊者仍然無法判斷

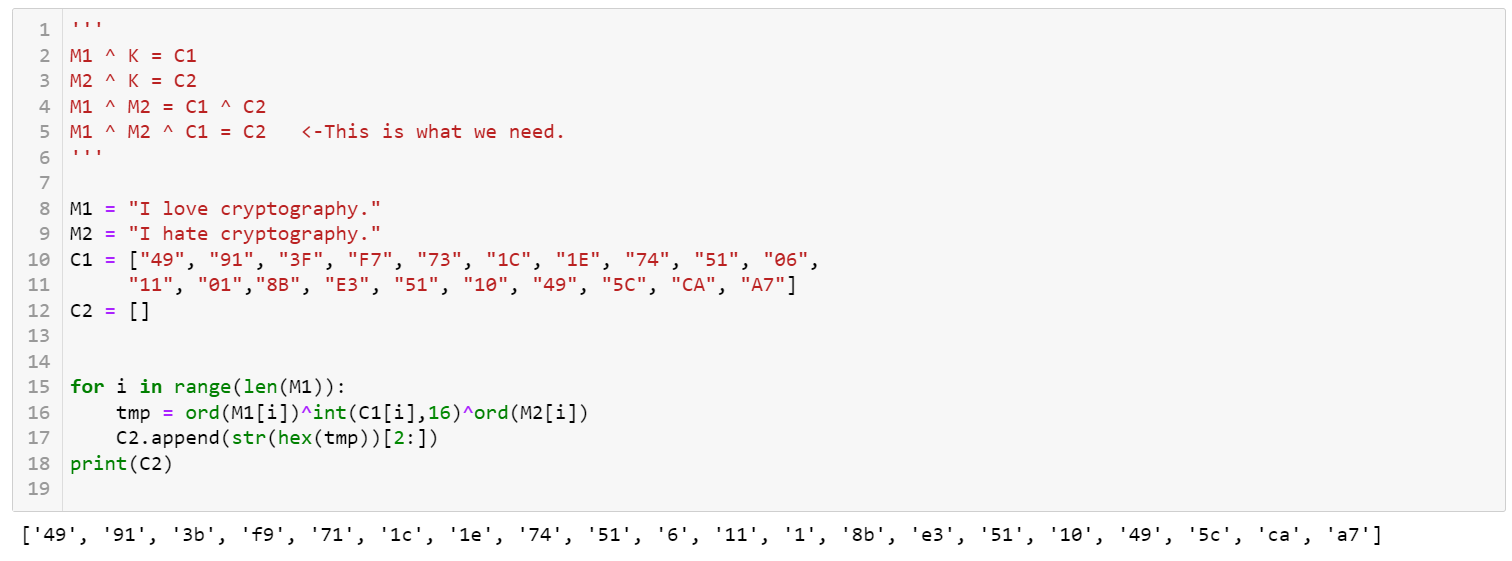
*P[Ek(m0) = c] ≈ P[Ek(m1) = c]*

 把m反轉後拿去加密

這**是語意安全**，因為把m反轉不會影響*E(k,m)*，加密出來的結果攻擊者仍然無法判斷

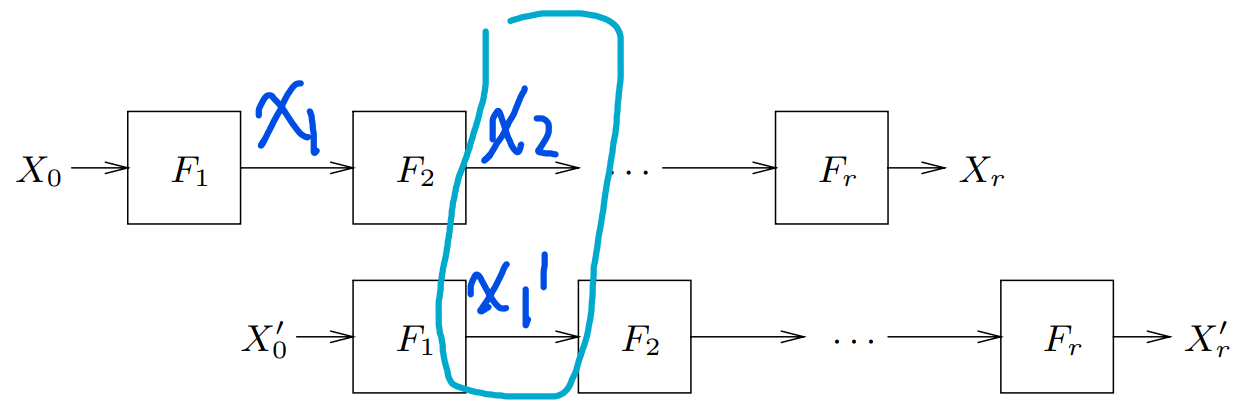
*P[Ek(m0) = c] ≈ P[Ek(m1) = c]*





I hate cryptography. -> 49913BF9711C1E74510611018BE35110495CCAA7



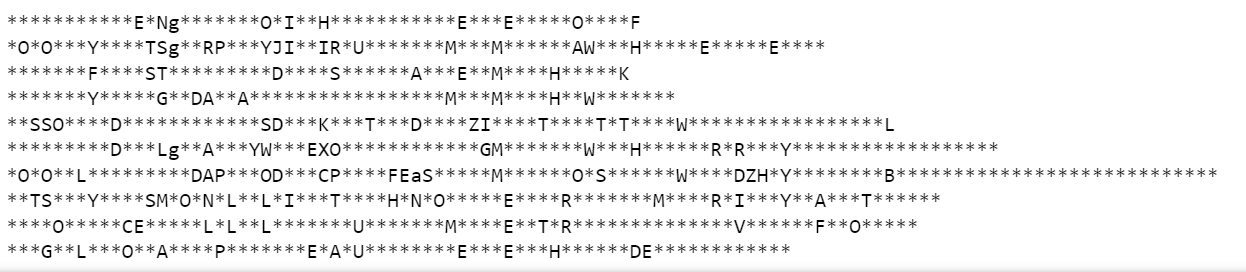


Slide Attack會被用在破解區塊加密法(Block cipher: 把明文分成很多塊，每塊要一樣長，然後把他們個別加密)，是指在使用相同輪函式(加密方法)的情況下，如果每個區塊的金鑰不互相獨立，差一個輪次的密文就可能會有關係。

攻擊者可以選擇兩個有部分相同的明文(x0和x0’)放進去跑加密，他們在一樣的輪次結束後，會得到部分相關的密文，如果不同輪次的金鑰有相關，兩個密文相關的部分就可能會傳下去(如果F1的金鑰和F2的金鑰有相關，因為在Feistel network encryption中F1和F2相同，所以X2會有部分和X1有關，又因為X1和X1’有關，X2就可能和X1’有關)，透過觀察相關的部分，就可能可以得到密鑰。



程式碼在OTP.ipynb



\*o\*\*only\*does\***god**\***play**\*dice\*\*out\*\*e\***some**z\*me\*\***throw**n\*them\*where\*they\*\*\*\*\*ot\*\*

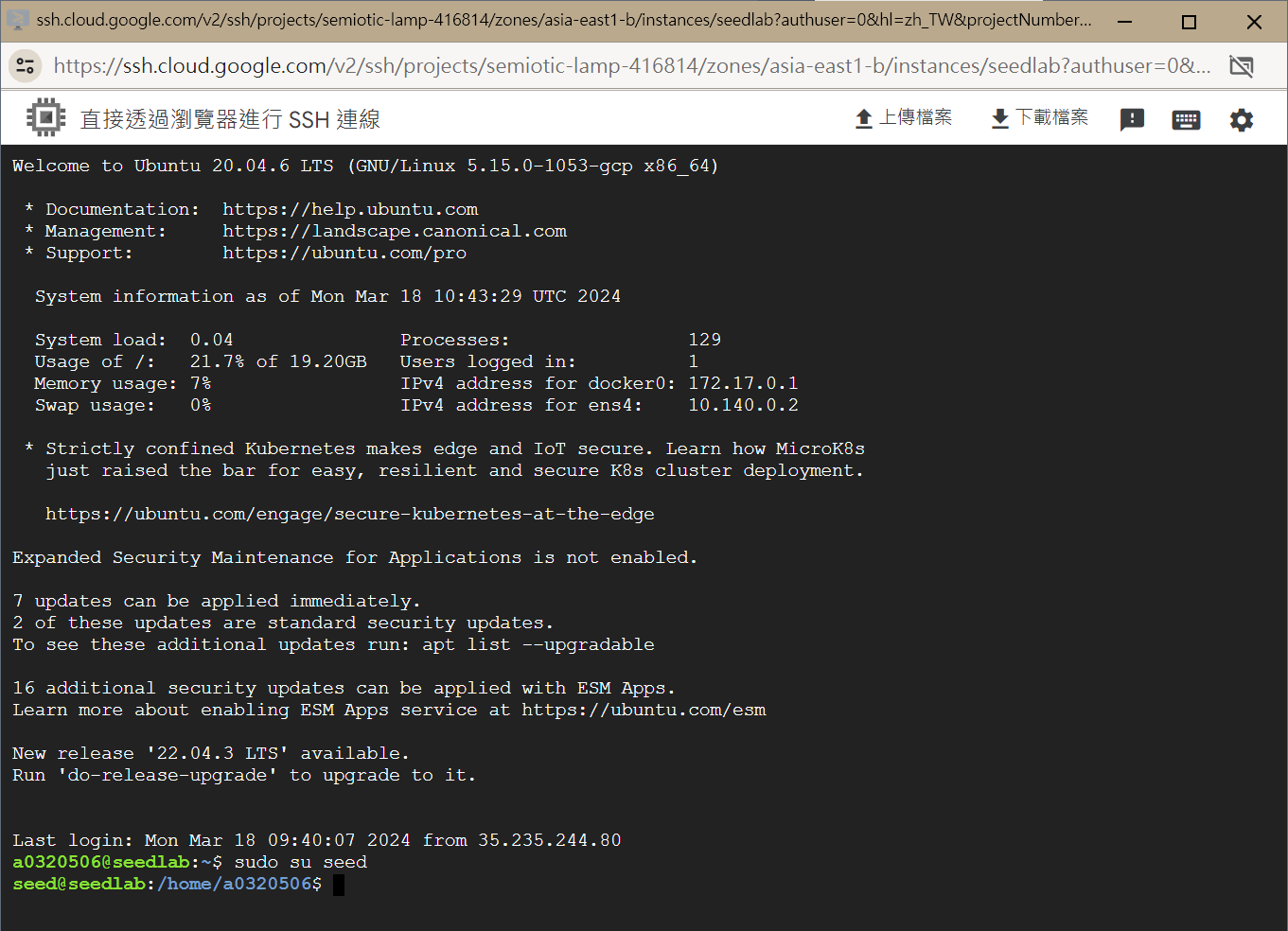
可能是空格的地方 [3, 8, 13, 17, 22, 27, 28, 32, 42, 45, 52, 57, 75]標成紅色，綠色是我自己湊的字，把上面那一行放到google，得到答案是

**Not only does God play dice, but he sometimes throws them where they cannot be seen**

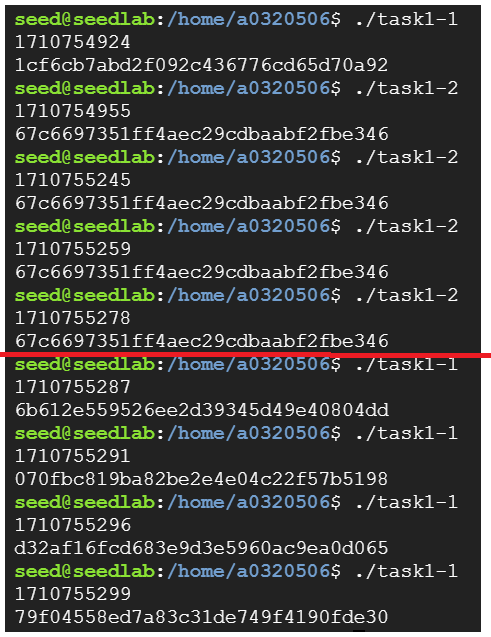


環境: Google Cloud





1.7.task1



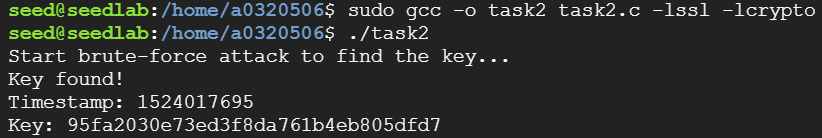
task1-1 : 原本的程式 task1-2 : 註解掉

task1-1和task1-2的差別:

**srand(time(NULL))** 是用來初始化亂數產生器，讓亂數種子是現在的時間，如果把它註解掉，就會產生一樣的key，因為種子不變亂數產生的結果就不變。

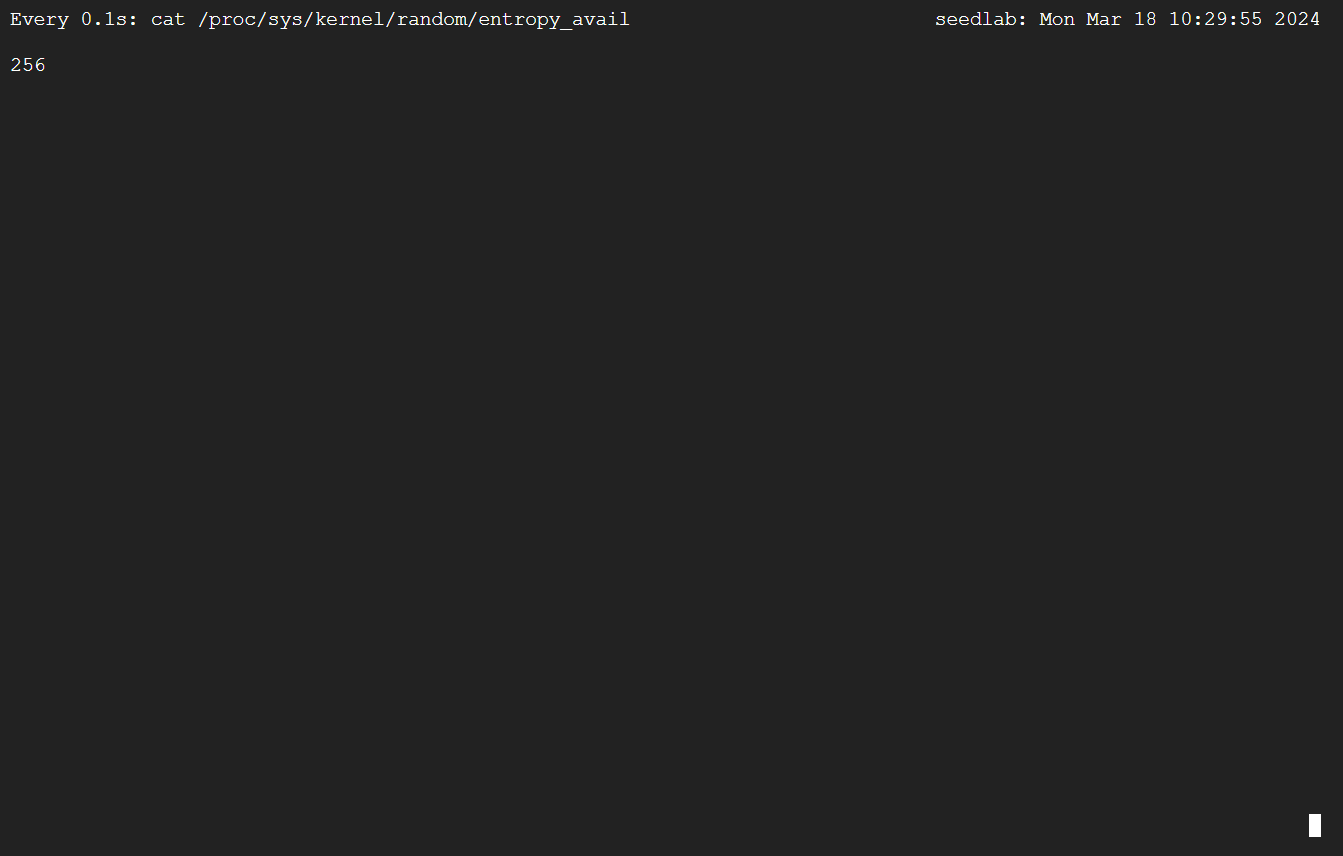
如果保留srand(time(NULL))，每次執行程式時亂樹種子就會更新，產生的key就會不一樣

1.7.task2



1.7.task3



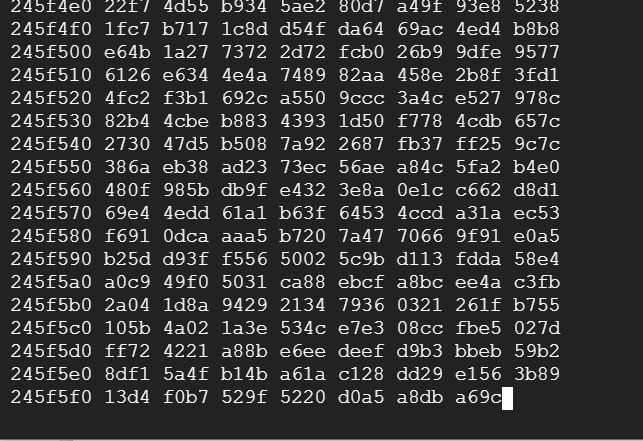
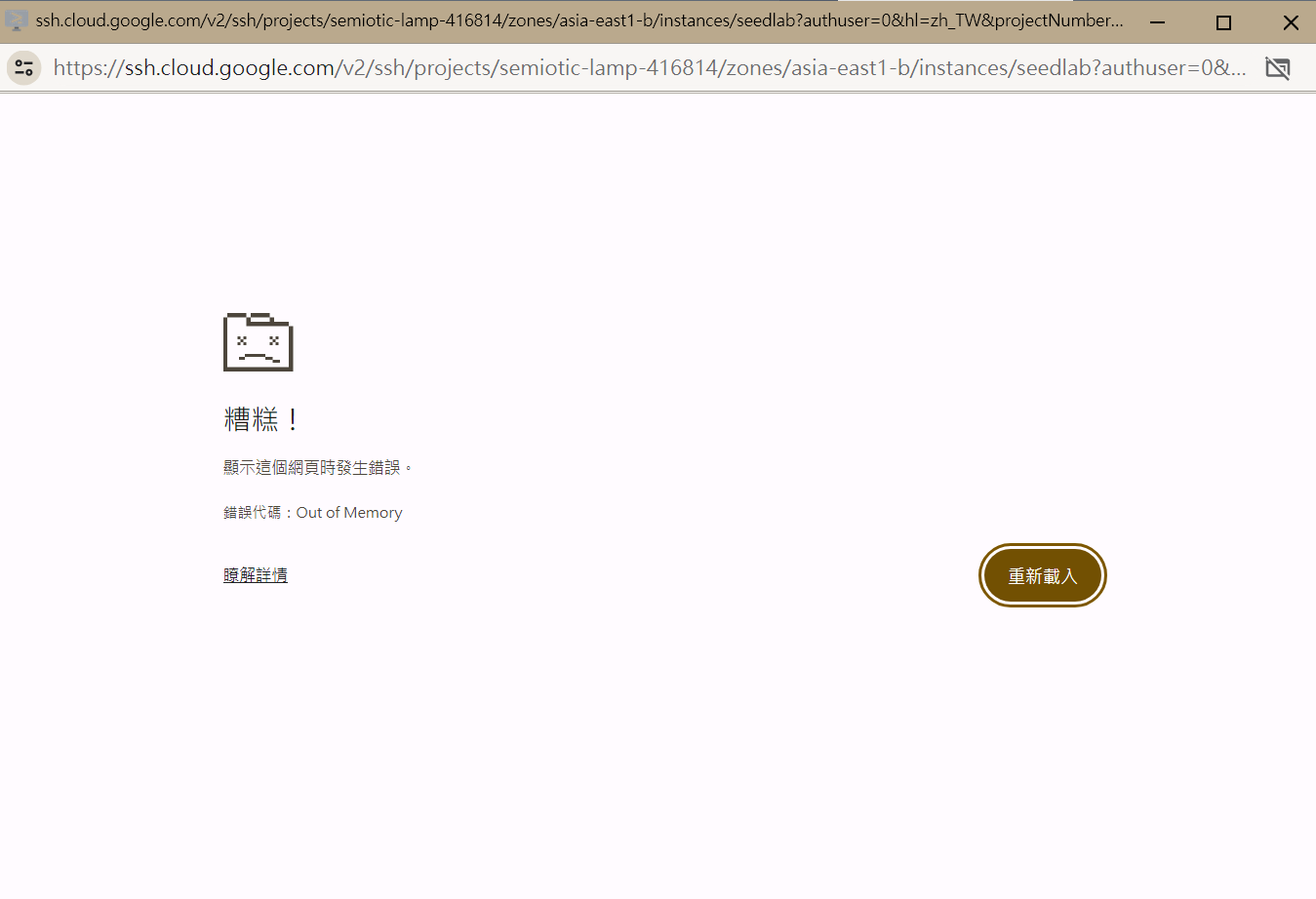


什麼都沒有發生

但理論上進行不可預測的操作會增加統中的熵，因為這讓系統中的隨機事件變多

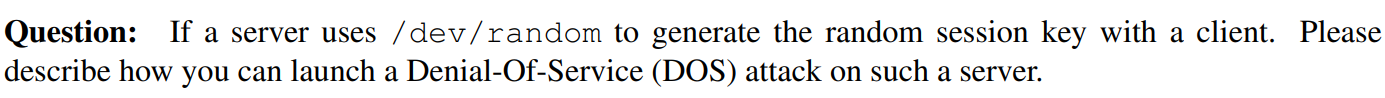
1.7.task4



放著跑五分鐘，就壞掉了

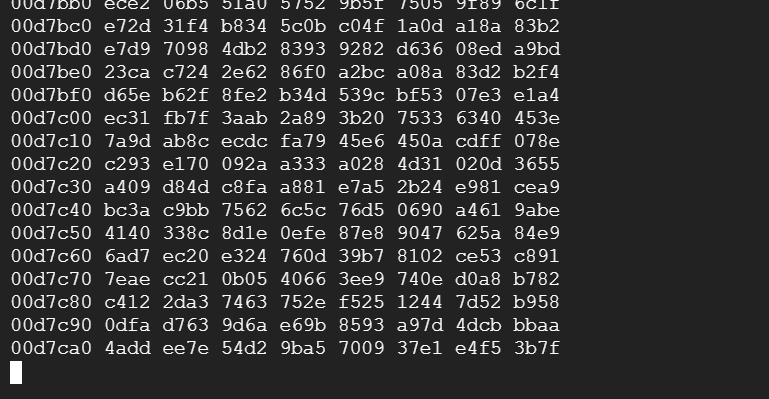
但理論上跑到最後就不會在出現新的隨機數，因為熵用完了，如果要讓他繼續產生隨機數，就要讓熵增加，可以透過滑鼠點擊之類的行為增加



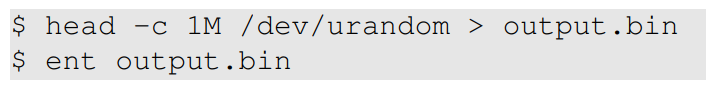
因為熵耗盡之後就不會產生隨機數了，要攻擊的時候可以發送大量請求把熵消耗掉

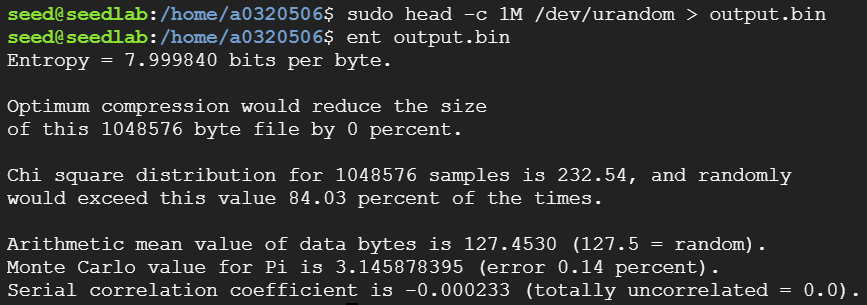
1.7.task5

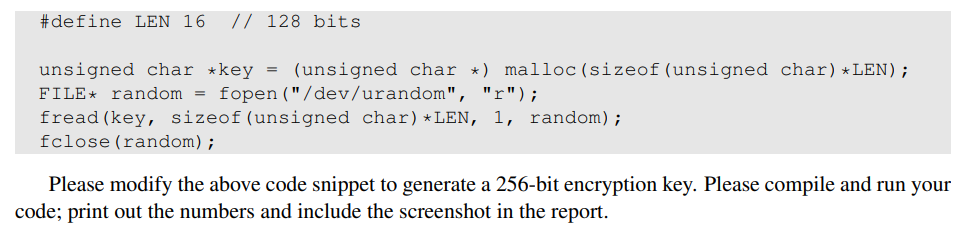


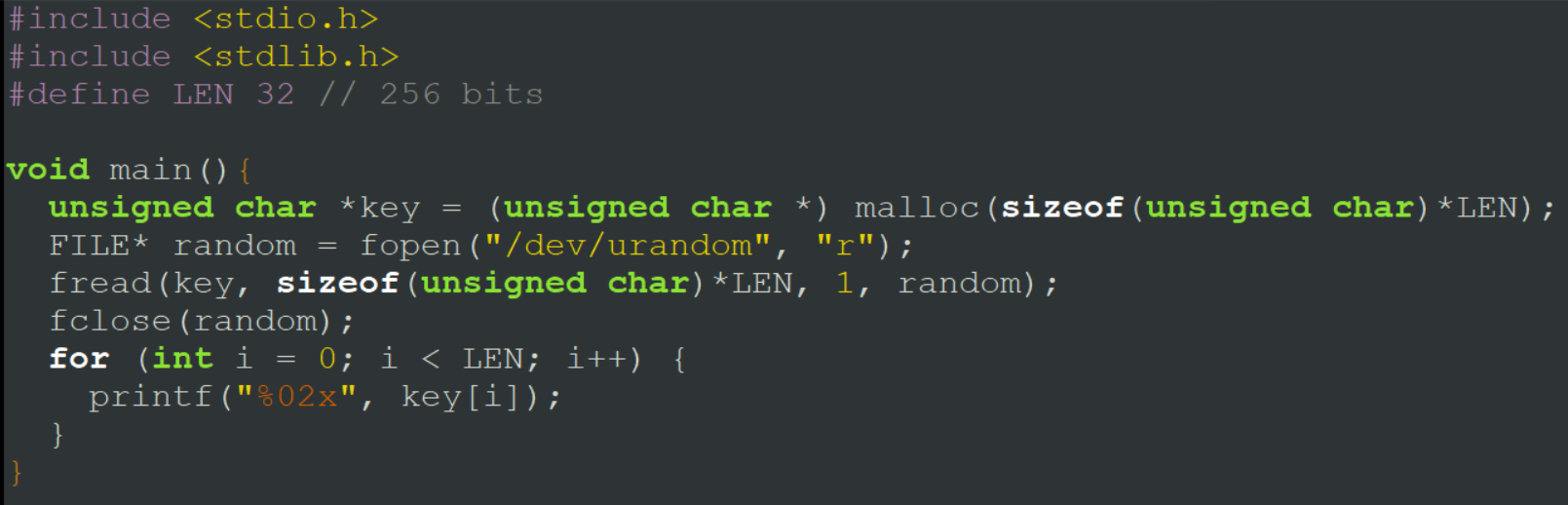


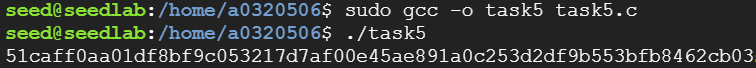
和task4不同，即使在熵不足的情況下，urandom也會持續產生新的隨機數

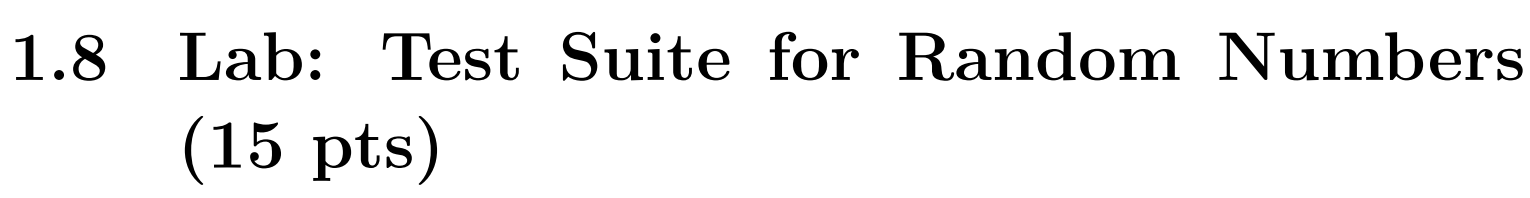








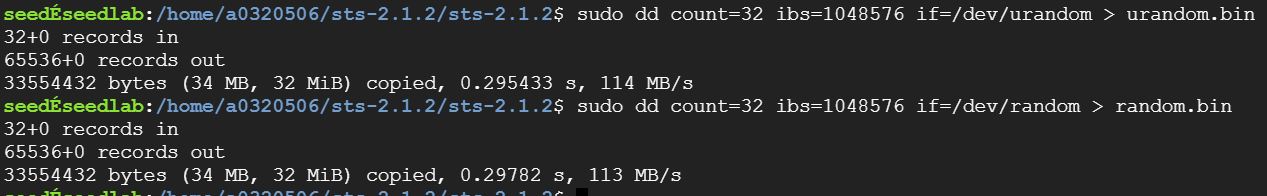


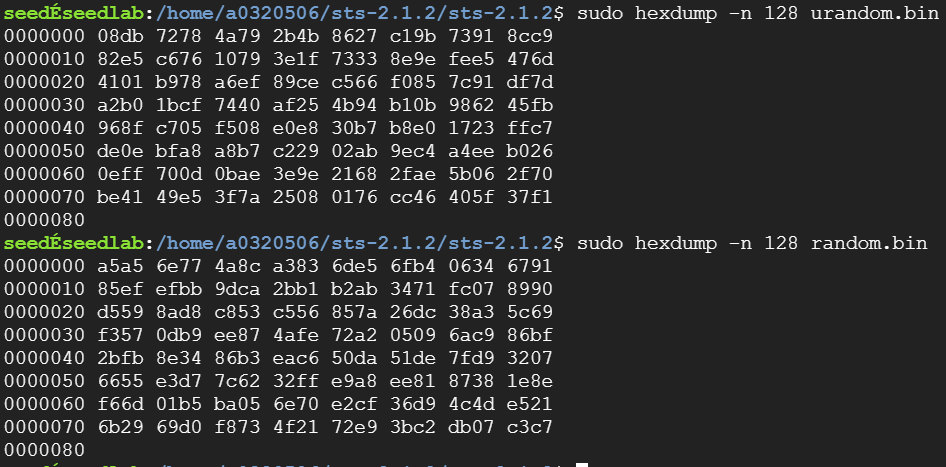


解壓縮檔案

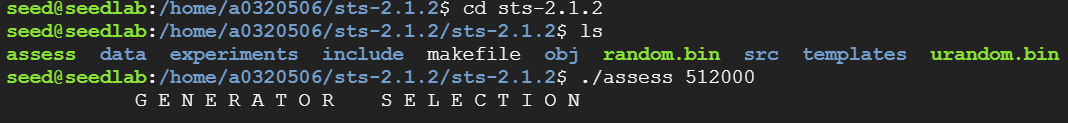


製作隨機數

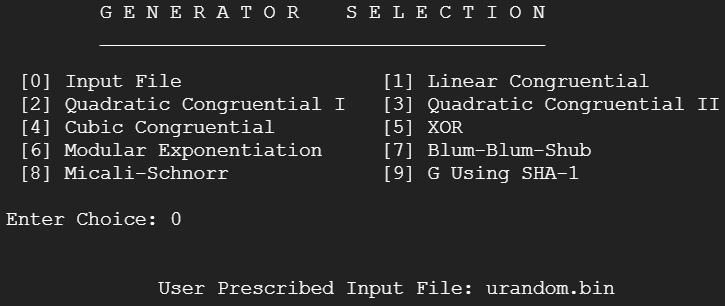




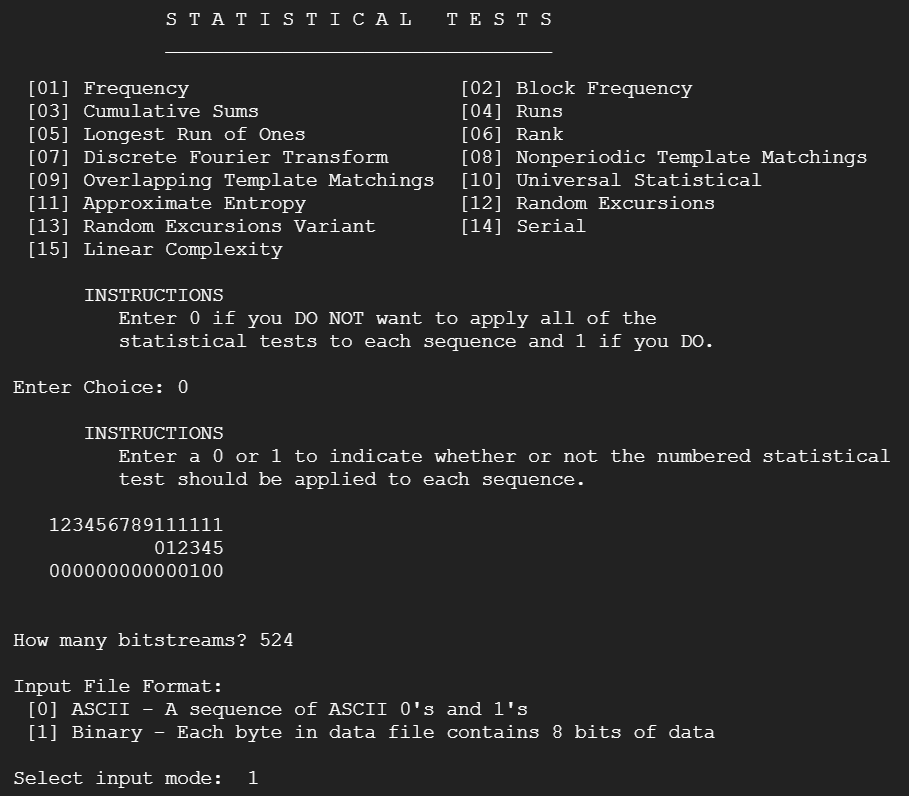
執行程式 (512000:一個序列有多少bit)



輸入要測試的檔案名稱

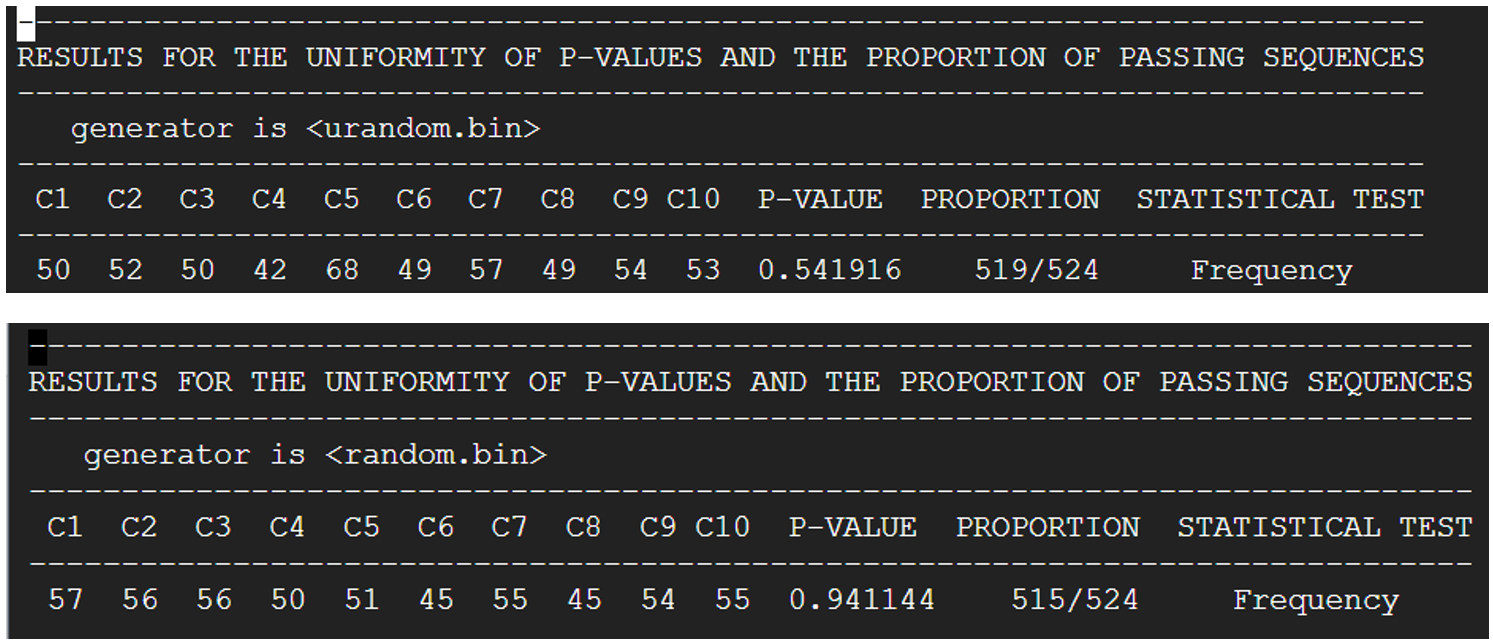


選擇要做的測試和輸入檔案有幾塊 (*(檔案大小/512000)\*8 無條件捨去*)

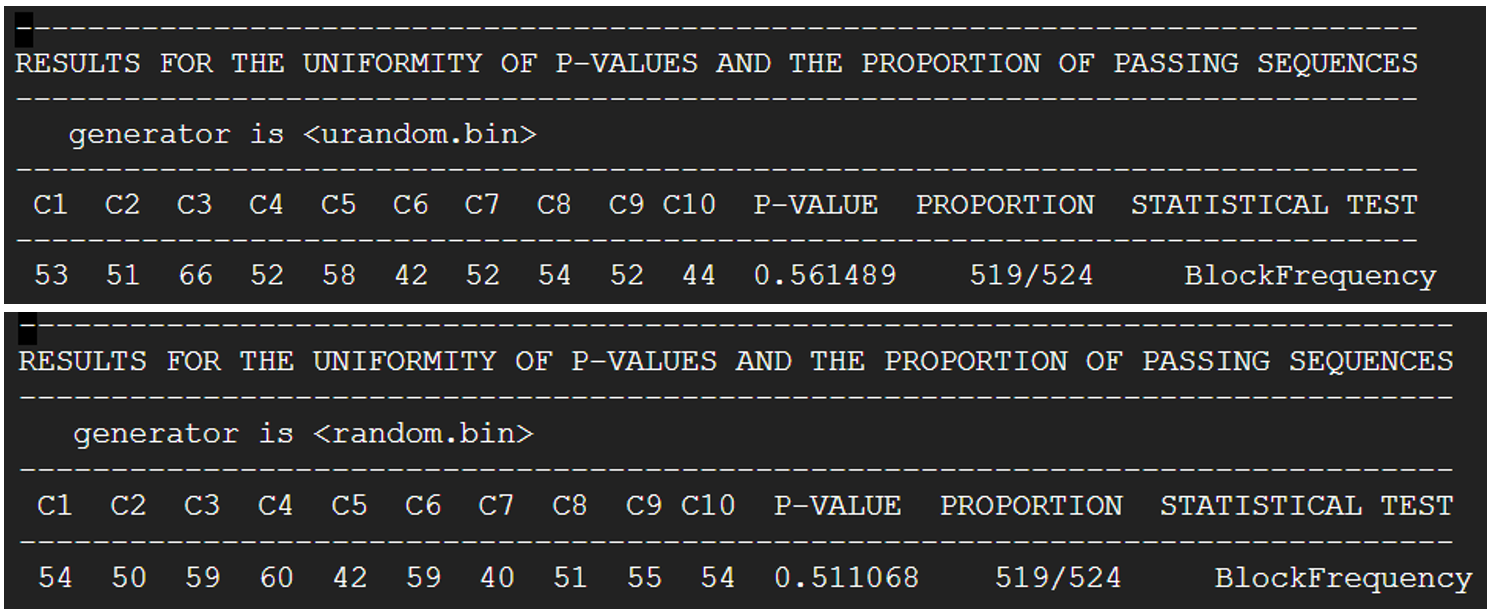


**P-VALUE > 0.05 : 測試通過**

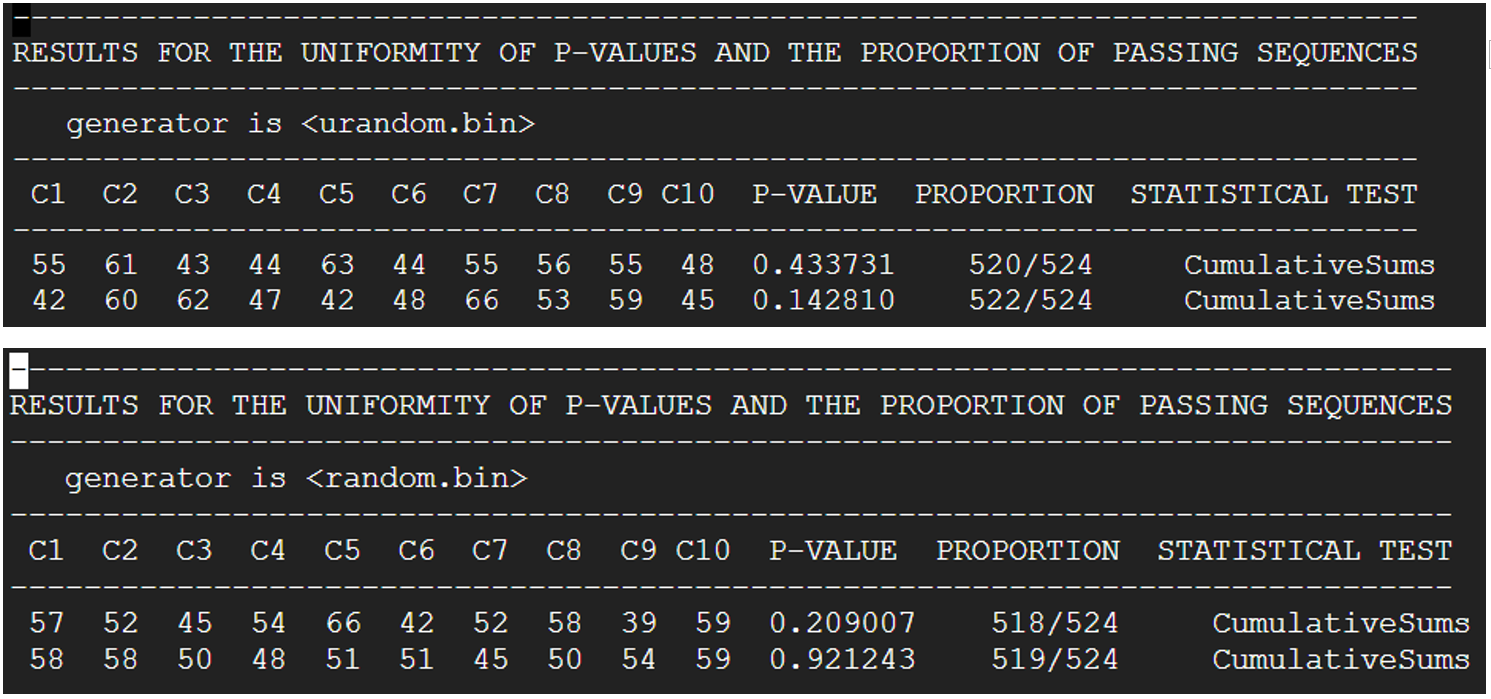
【Frequency】看0和1在整個序列中所佔的比例，確定序列中的1和0數是否與真正的隨機序列中的1和0數近似相同，若通過該測試，則被測序列是隨機的



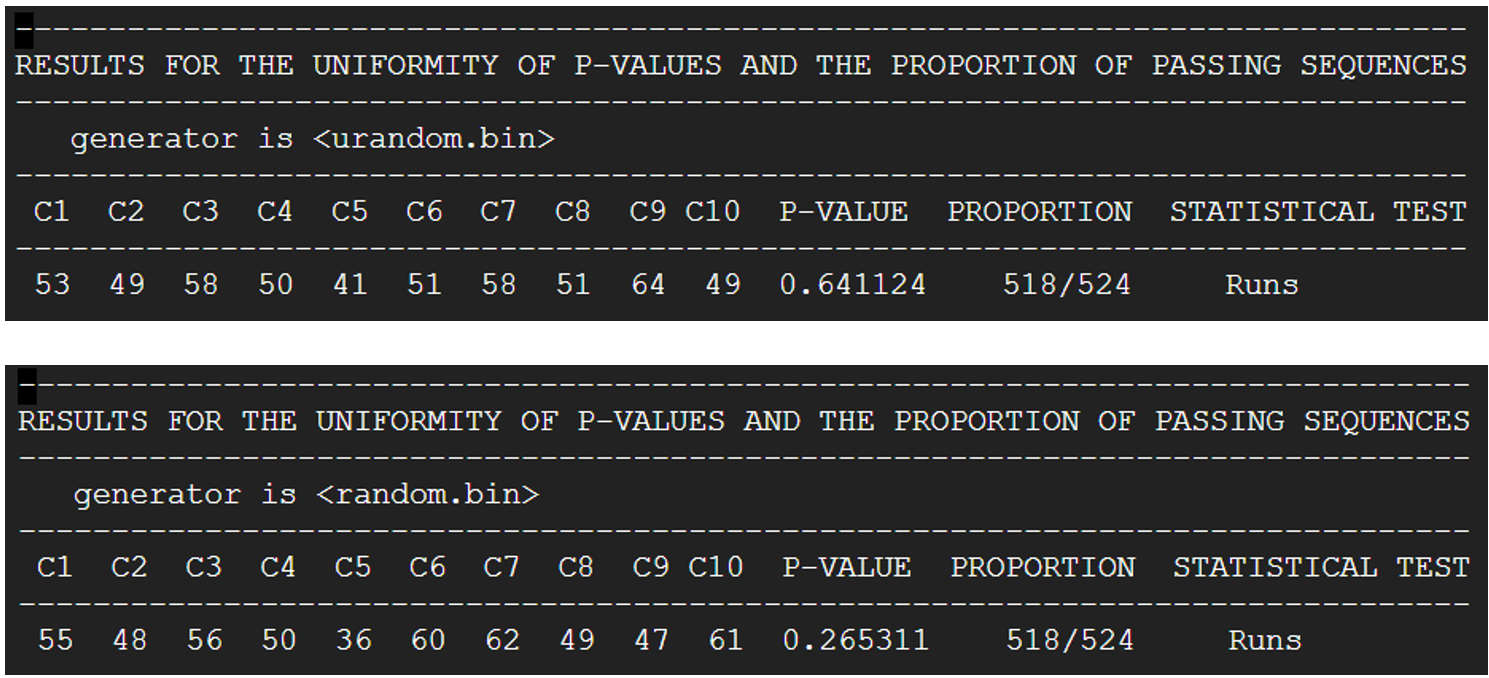
【Block Frequency】一個特定的區塊值在給定的數據序列中出現的次數



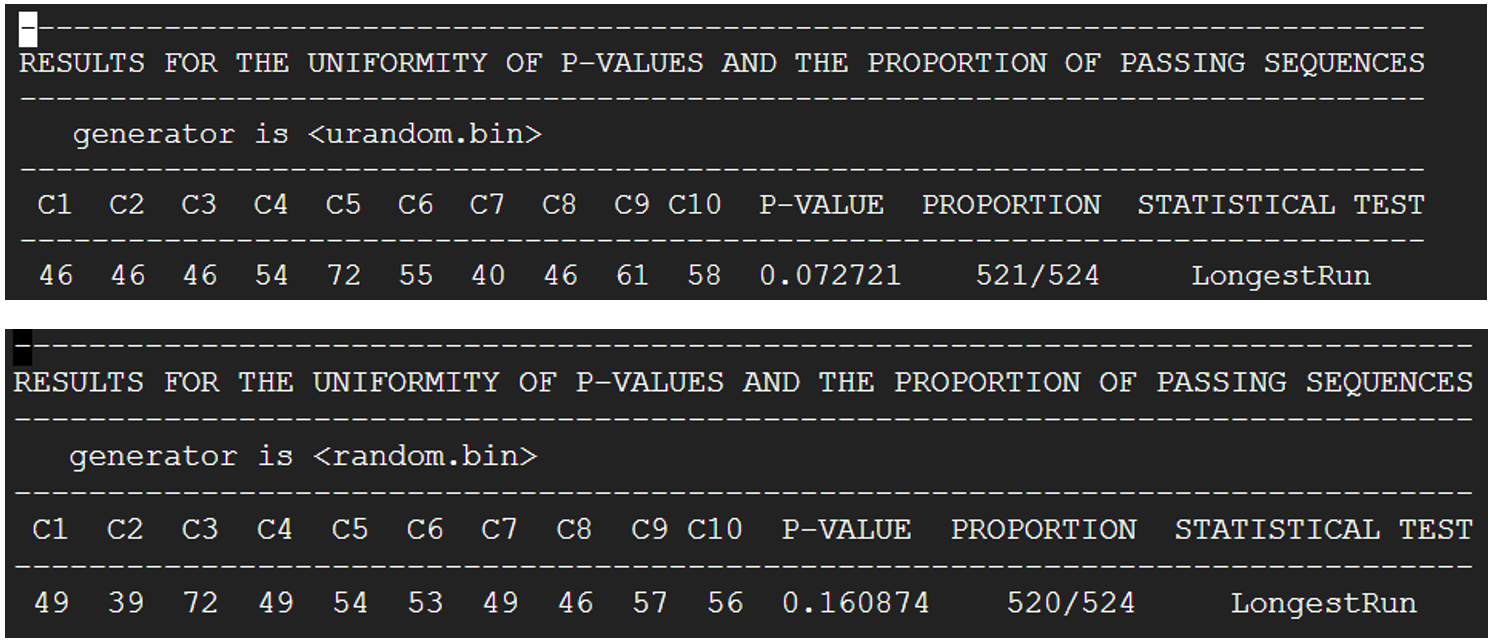
【Cumulative Sums】將一個數據序列的元素進行累積求和，然後計算這些部分和的最大偏離程度。如果數據序列是隨機的，那麼這些部分和的最大偏離程度應該在一個合理的範圍內



【Runs】計算序列中出現的連續相同值的次數，稱為 "runs"。然後，計算實際的 "runs" 的數量，以及根據隨機性期望的 "runs" 的數量。這兩個數字之間的差異用於評估序列是否表現出非隨機的特性



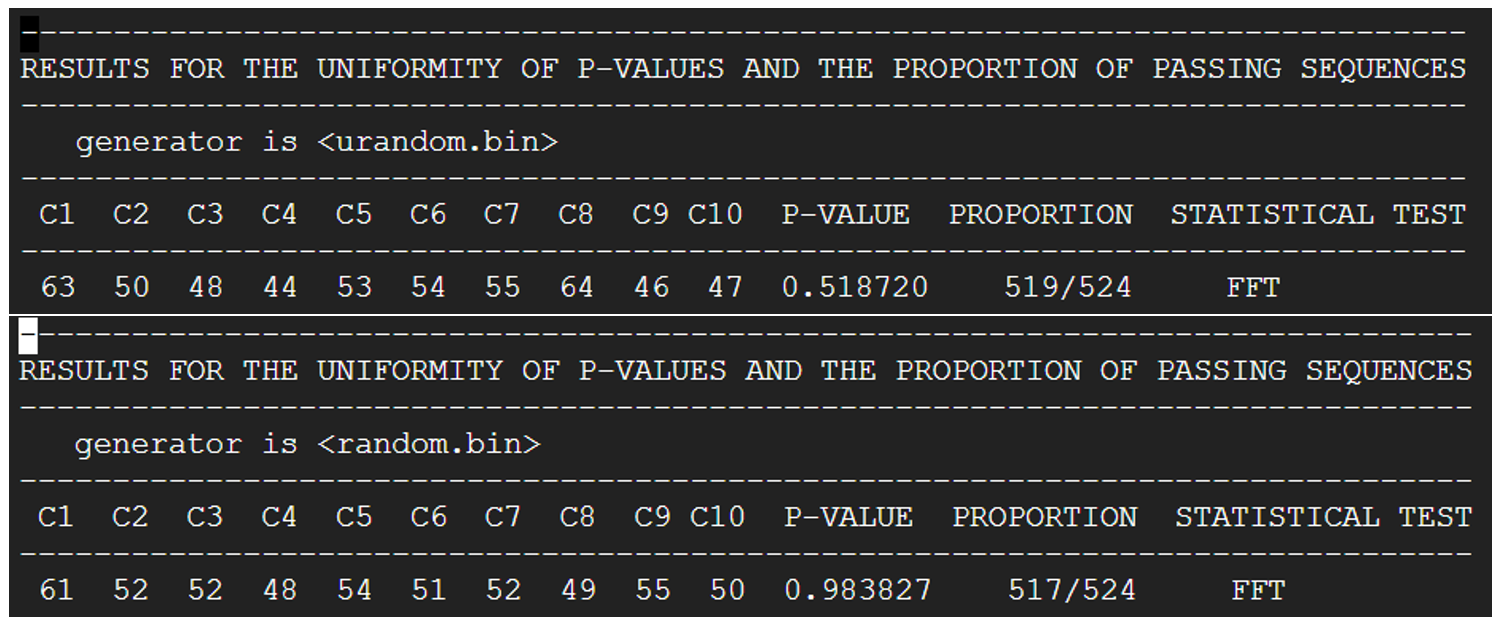
【Longest Run of Ones】評估一個二元序列中連續出現的最長 "1" 的序列是否符合預期的隨機性



【Rank】評估矩陣中的線性獨立性



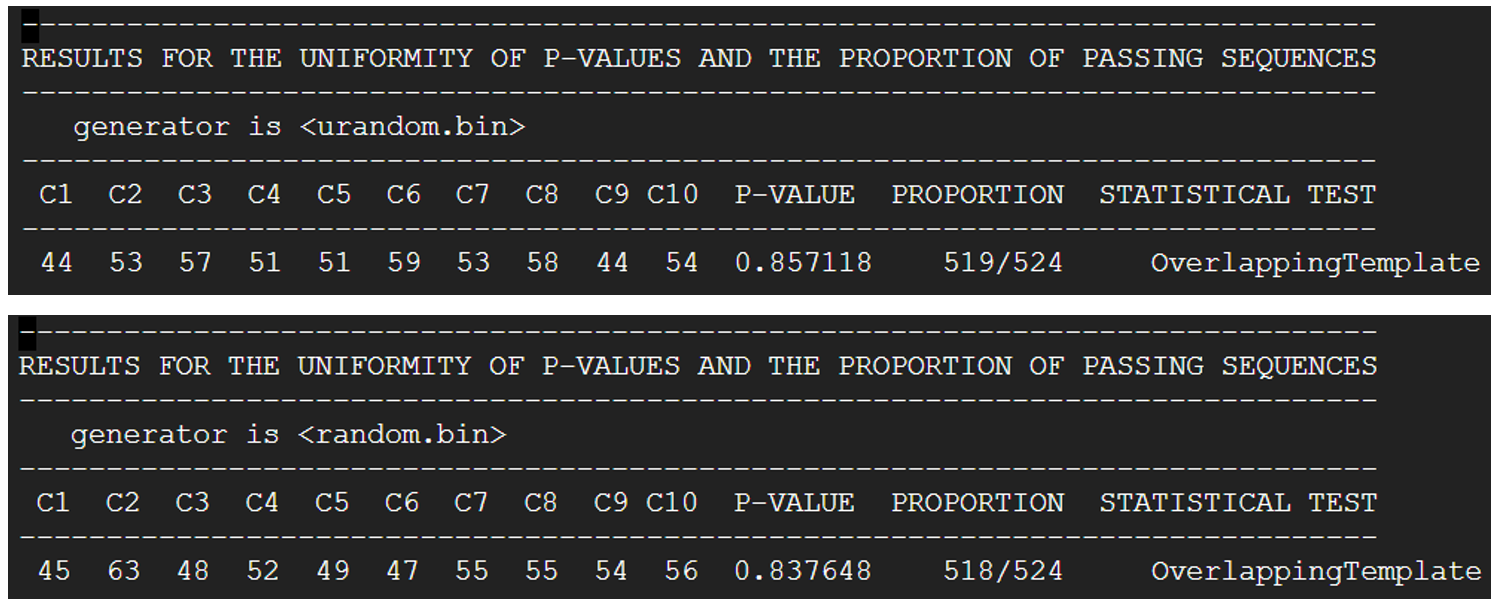
【Discrete Fourier Transform】將數據序列轉換為頻率表示，以檢測是否存在特定的頻率成分



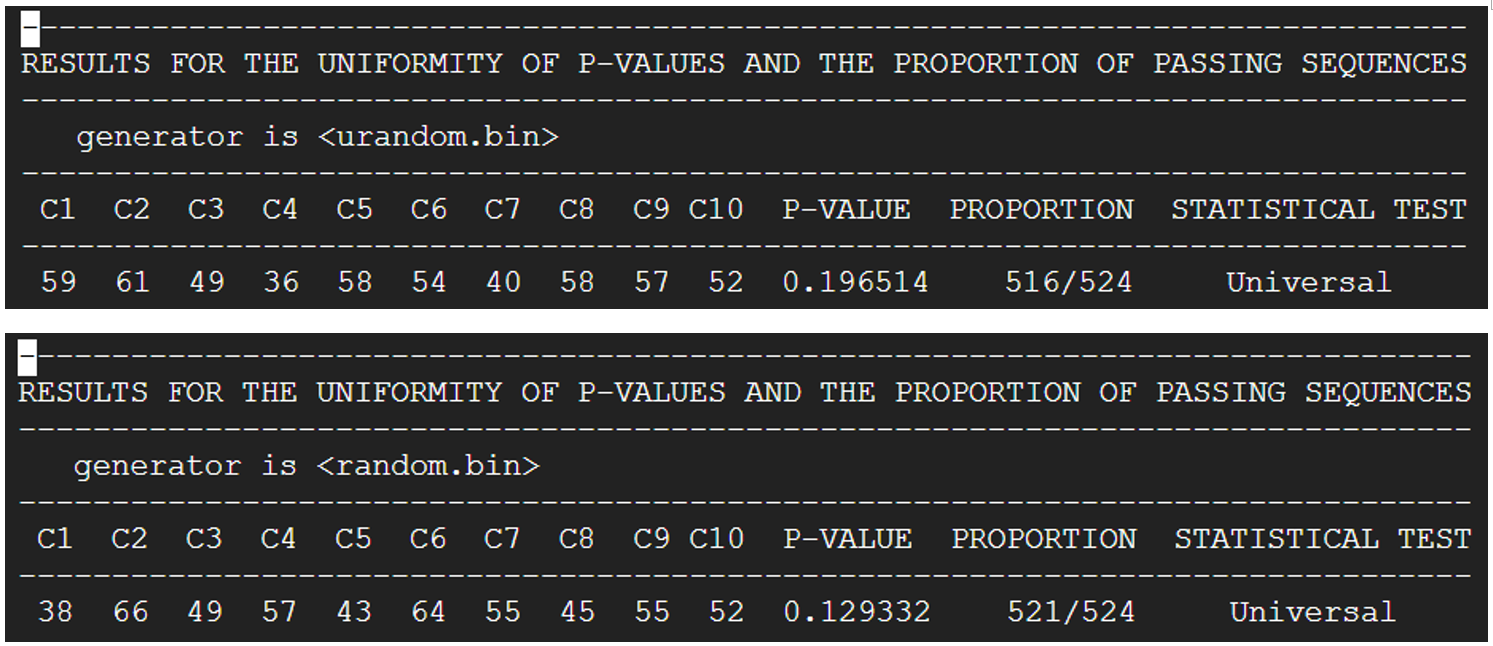
~~[08] Nonperiodic Template Matchings 跑不出來~~

檢測數據序列中是否存在與給定模板相同的子序列

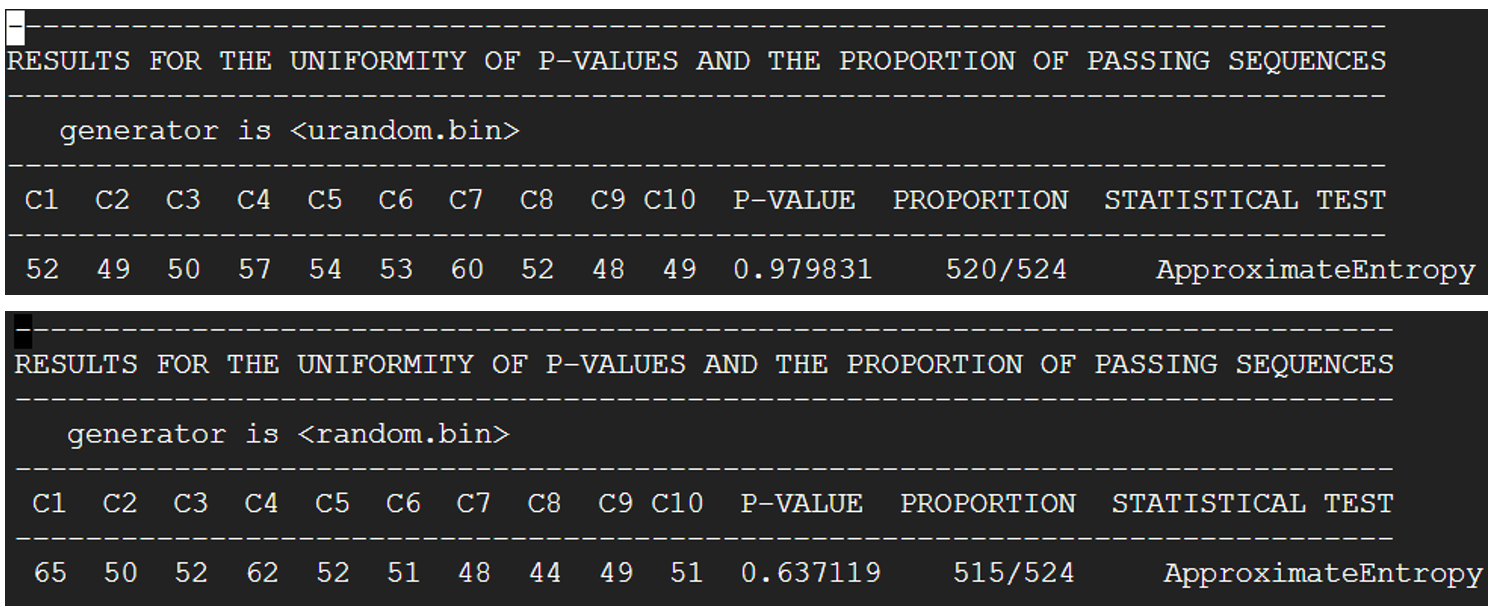
【Overlapping Template Matchings】類似於Nonperiodic Template Matchings，但它允許模板在序列中重疊出現



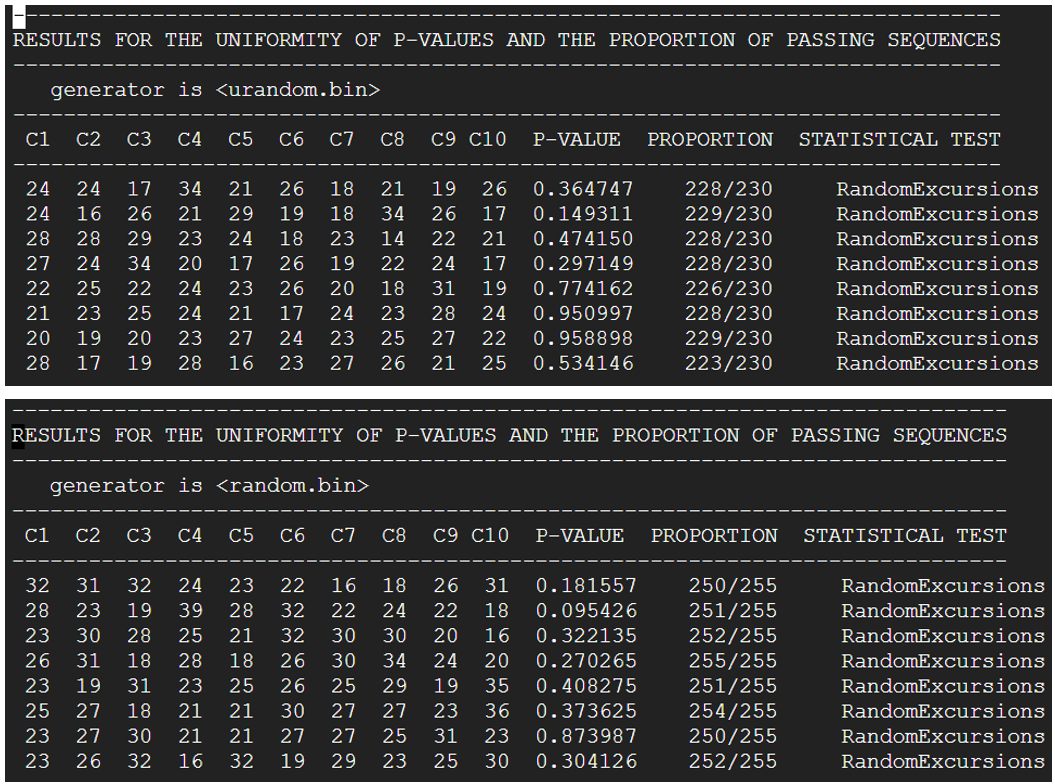
【Universal Statistical】利用統計學原理來評估數據序列的隨機性



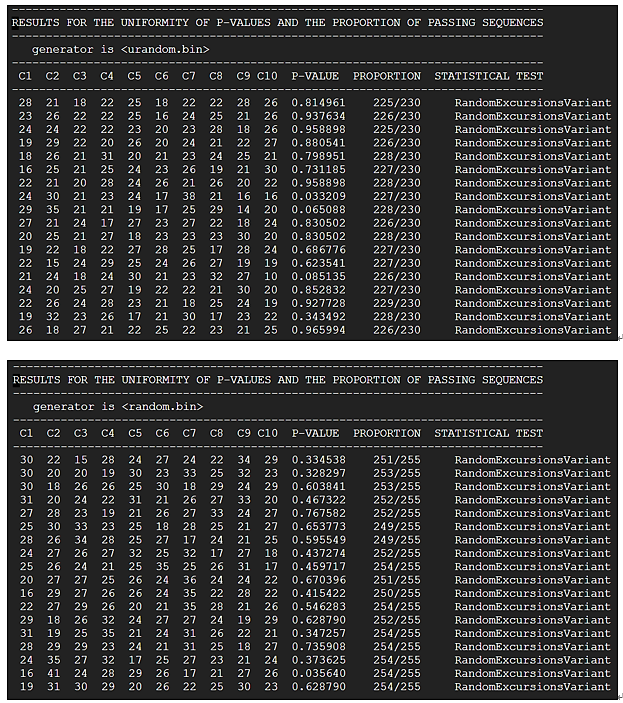
【Approximate Entropy】測量數據序列的複雜性，以檢測序列中的規律性和隨機性



【Random Excursions】檢測在隨機序列中出現的偏移或異常



【Random Excursions Variant】與Random Excursions類似，但在計算過程中考慮了特定的變異，以提高檢測的敏感性



~~[14] Serial 跑不出來~~

檢測數據序列中是否存在組合模式或相關性

~~[15] Linear Complexity 跑不出來~~

測量數據序列的線性複雜性