ADA

Kon Yi

September 4, 2025

Abstract

The lecture notes of ADA of 呂學一.

Contents

Chapter 1

為什麼要設計跟分析演算法

1.1 一些例子說明設計跟分析的重要性

Lecture 1

4 Sep. 14:20

設計演算法的切入點很重要，舉個例子：假設有 500 毫升的咖啡及 500 毫升的牛奶，假設用一個 10 毫升的湯匙撈咖啡放到鮮奶裡，接著由 510 毫升的咖啡牛奶用同個湯匙撈 10 毫升回剩下 490 毫升的咖啡那杯，請問牛奶咖啡裡牛奶的比例還是咖啡牛奶裡咖啡的比例比較少？事實上不論有沒有攪拌均勻最後的濃度都一樣，因為最後兩杯都是 500 毫升，所以假設咖啡牛奶有 x 毫升的咖啡那就有 $500-x$ 毫升的牛奶，也就代表另外一杯會有 $500-x$ 毫升的咖啡以及 x 毫升的牛奶因為牛奶跟咖啡的總量都是 500 毫升。這個例子告訴我們切入點很重要。

再看下一個例子，請問如果要把一副撲克牌洗亂平均要洗幾次？首先我們會需要定義亂的定義，我們可以定義成在洗完牌後 52! 種排序出現的機率都相等。現在我們規定都要用 riffle shuffle 這種方式，那如果用演算法分析就會知道洗 7 次是充分且必要的（這是某個 paper 的結論）。

現在如果我們考慮另一種洗牌方法，叫做 top-in shuffle，這種方法每次洗牌都把最上面那一張用等機率的機率分布插入到剩下 51 張牌產生的 52 個空隙，我們希望找出一個 n 使得在 n 次 top-in shuffle 後每一種排列出現的方式都是 $\frac{1}{52!}$。現在我們叫開始第一次洗牌前最下面的那一張牌底牌，並定義階段數 k 代表底牌現在在從底下往上數第 k 張，所以一所以一開始是第一階段。

Theorem 1.1.1. 雖然不知道其他牌怎麼樣，但是在任何時間點底牌下方的牌一定是亂的。

Proof. 我們可以用數學歸納法，假設做 t 次 top-in shuffle 後底牌下面的牌的排列叫 p_t，那我們易知 p_0 跟 p_1 都是亂的，現在假設 p_n 是亂的，那因為下一次 shuffle 後任意排列的機率都一樣，所以 p_{n+1} 也是亂的。 ■

現在有這個底牌的觀點後，我們可以知道在到達第 52 階段時還不夠亂，但只要再做一次 shuffle 就能保證所有牌是亂的，現在我們想算從第 k 階段到第 $k+1$ 階段會需要多少次，我們可以知道平均要 $\frac{52}{k}$ 次因為最上面的牌只要插到 52 個空隙裡的 k 個就會使階段數上升，所以平均會需要

$$\sum_{i=1}^{52}\frac{52}{i}=52\cdot\left(\frac{1}{1}+\frac{1}{2}+\cdots+\frac{1}{52}\right)=52\cdot H_{52}$$

次。現在我們要思考也許這不是必要條件，也就是其實還有更好的方法可以用更聰明的解法得到更小的期望值。

現在我們同樣用底牌的概念切入，也許有更好的方法能得到更小的期望值？其實如果你在一開始拿到牌時選第二張當底牌，那其實第二張下面的牌就已經是亂的，所以只要再洗一次牌就好了。這個例子告訴我們即使是同個切入點（設計），分析的能力也會影響研究的結果。

1.2 演算法的一些行話

Definition 1.2.1 (問題 Problem). 一個問題是指一個給訂合法 input 都有 output 且待解決的事物

解問題的種類：

- 幫一個問題找到一個快速的解法。
- 如果找不到，可能是這問題太難了，但也可能是我們不夠厲害。
- 因此我們需要學習某些問題是否是真的很難還是是我們不夠厲害。

Definition 1.2.2 (問題個例 Problem Instance)**.** 給訂一個問題且給定 input，並且我們要找到這個 input 在這個問題對應到的 output，這樣叫做一個問題個例。

Definition 1.2.3 (計算模型 Computation Model)**.** 演算法的遊戲規則叫做計算模型，同個問題在不同模型下可能有不同的難度。

Definition 1.2.4 (演算法 Algorithm)**.** 一個詳細的有步驟的指示，只要符合遊戲規則則沒有規定其形式包含詳細程度還有使用的語言之類的。

解決一個問題的演算法就是解決該問題的解題步驟。

Note. If an algorithm produces a correct output for any instance of the problem, we say that this algorithm solves the problem and we say that the algorithm is a (correct) algorithm for the problem.

Definition 1.2.5 (困難程度)**.** 就是一個問題有多難解。

Example. 如果用魔方的旋轉次數來衡量問題難度，那對於任何解法都至少有一種起始狀態會要轉 20 次才會恢復而且對於任何初始狀態都有一種解法可以 20 轉把魔方恢復，也就是說魔方問題已經被完全解決因為他已經被最佳化。

Note. 在這門課的估計都是做 worst case 的估計。

Note. 要用 upper bound 跟 lower bound 去壓一個問題直到 upper bound 等於 lower bound 我們才會說這個問題解決了。

Lecture 2

11 Sep. 14:20

Appendix