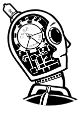
簡介

即使您認為自己對程式語言已很了解，但在編寫程式時怎麼還是會卡住呢？您是否讀懂了某本程式語言參考書中的某個章節，卻無法把自己所讀到的東西應用在實際的程式上？您是否有理解了網路上看到的某個程式範例，甚至每行程式所做的事情都能清楚明白，但自己在面對實際的程式設計工作時，卻只能腦筋一片空白地對著文字編輯器的空畫面發呆。

不是只有您有上述這樣的情況。我教授程式設計已超過 15 年了，大多數的學生在學習的過程中，某種程度都有碰到上述的這些情況。這是因為缺少了*問題解決*（*problem solving*）的能力，這項技能可以讓您在面對一項問題描述後，設計編寫出程式來解決這項問題。不是所有的程式設計工作都需要廣泛的問題解決能力，如果只是對現有的程式進行小幅度修改、除錯或新增測試程式碼，這類程式設計工作在本質是比較機械式的，不會在創造力上有什麼挑戰性。但程式總該是用來解決問題的，所以優秀一流的程式設計師都需要具備解決問題的能力。

碁峯
www.gotop.com.tw



問題解決（problem solving）並不容易，但有些天才卻把這項能力變得很容易，就像 Michael Jordan 在運動上的天賦。對於這類天才，高階的思維能很輕鬆地轉換成原始程式碼。以 Java 來比喻，他們的腦子天生就能直接執行 Java 程式碼，而我們大多數人只能透過虛擬機器來直譯和執行。

沒有超人的天賦對於我們成為程式設計師並不會有什麼重大的影響，如果真有影響，那世界上就沒有多少程式設計師了。我已經看到太多有潛質的學習者在沮喪中掙扎太久。在最壞的情況下，他們完全放棄程式設計這條路，認定自己永遠不可能成為程式設計師，而覺得只有天賦異稟的人才能成為優秀的程式設計師。

為什麼學習解決程式設計問題這麼困難呢？

部分原因歸咎於解決問題與學習程式語法是不同的行為，因為它用了不同的心理「肌肉」。學習程式語法、閱讀程式、記憶 API 的元素等，這些都屬於分析性的「左腦」活動。運用之前學會的工具和技能來編寫原始程式碼則是一項創造性的「右腦」活動。

假設您想要拿出掉入屋頂雨水槽中的樹枝，但梯子的長度卻不夠長，讓您拿不到樹枝。這時您會到車庫找些輔助工具來協助我們取下樹枝。看看有沒有什麼辦法可讓梯子加長？或是站在梯子上時能不能手持某樣工具來捉取樹枝？或許也可以從別處爬上屋頂來拿出樹枝。以上這些解決問題的過程就是一項創造性的行為。不管您信不信，在設計自己原創程式時的思維過程與想辦法從屋頂雨水槽中拿出樹枝的思維過程是非常相似的，但與 `for` 迴圈除錯的思維過程則完全不同。

不過，大多數的程式設計書籍都把重心放在語法和語意上。雖然學習程式語言的語法和語意是不可少的，但這只是學習如何使用該語言來設計程式的第一步而已。從本質來看，大多數以初學者為導向的程式設計書籍都只教怎麼閱讀程式，而不是如何設計編寫程式。另外那些專注於怎麼編寫程式的書籍則通常只是有效的「食譜」而已，因為這類書只會教特定的「作法」，供應您在特定情況下使用。這類書能節省時間，但對於學習設計編寫原創程式卻沒什麼價值。請以原創食譜的角度來思考，雖然每位一流廚師都有自己的食譜，但沒有人只靠食譜就成為一流的廚師。優秀的廚師會了解各種食材、備料和烹飪方法，並且知道怎麼把它們組合起來烹煮出美味的佳肴。一流廚師在烹飪時所需要的就

碁峰 www.gotop.com.tw

是一間設備充足的廚房。同理來說，一流的程式設計師了解程式語法、應用程式框架、演算法和軟體工程的原理，並且知道怎麼把它們組合起來製作出色的程式。提供優秀的程式設計師一份需求規格清單，讓他在功能齊備的程式開發環境中創作，他就能製作出優秀的作品。

一般來說，目前的程式設計教育在解決問題方面沒有提供太多指導。反而是假設程式設計師在有了程式開發工具並設計編寫夠多的程式後，他們最終將學會開發這類程式而且會設計編寫得很好。這種說法雖有理，但這個「最終」可能是要花很長一段時間。從啟蒙學步到初探入門的過程中可能充滿挫敗，有太多人信心滿滿地開始，卻在半路就放棄而未達目的地。

除了以反覆試誤的方式來學習，我們還可以用有系統的方式來學會解決問題的方法。本書的重點就是這個，我們可以學到組織自己想法的技術、發覺解決方案的過程、以及對某類問題所採取的策略。透過研究這些方法，我們就能發揮創造力。別搞錯了：程式設計，尤其是解決問題，是一項創造性的活動。創造力是很神秘的，沒有人能準確說出創造力的思維方式。但是，如果我們能夠學會音樂創作、體會文學創作的建議、或學到如何繪畫，那麼我們也能學會怎麼以創造力來解決程式設計的問題。本書不會精確地告知要怎麼做，但會協助讀者開發問題解決能力的天賦，讓您知道應該怎麼做。本書的目標是協助讀者成為自己心中所想的程式設計師。

我的目標是讓本書的讀者學會如何以系統化的方式完成每項程式設計工作，並相信自己最終能解決問題。當讀者學完本書時，我希望您能像程式設計師這樣思考，並相信自己已經是位程式設計師了。

碁峯 www.gotop.com.tw

關於本書

在說明了本書的必要性之後，我還會解說一下本書討論的內容，以及本書所沒有的內容。

先決條件

本書假設讀者已熟悉 C++ 的基本語法和語意，也已編寫過一些程式。本書大部分的章節都假定讀者具備了特定的 C++ 基礎知識。這些章節會從這些基礎的回顧開始，如果讀者還是程式語言的新手，也不必擔心，有很多 C++ 語法的書籍可參考，學習解決問題與學習程式語法是可以同時並行的。不過在嘗試每章後面的習題之前，要確定已學會了相關的程式語法即可。

書中所選的主題

本書所選的主題是我最常看到的程式新手會苦苦掙扎的領域，這些也是在初階和中階程式設計中會談到的廣泛跨領域主題。

不過我要強調一點，本書並不是一本用來解決特定問題的演算法或模式的食譜式書籍。雖然本書後半部的章節有討論如何運用眾所周知的演算法或模式，但不要把這本書當成「小抄」來克服特定的問題，也不要只關心與自己面臨的難題相關的章節。相反地，除了暫時先跳過因為缺乏學習先決條件而無法讀懂的內容外，讀者最好是逐章通讀整本書。

程式設計風格

這裡快速提示一下本書所用的程式設計風格：本書不會要求高效率程式設計成果，也不要求最簡短、高效率的程式碼。書中程式碼所用的風格是以容易閱讀為首要考量因素。在某些情況下，我會用多個步驟來完成某些只需一個步驟即可完成的工作，其原因是為了更清楚地展示說明其原理。

本書也會討論程式設計風格的某些內容，但僅是針對較大的議題部分，例如類別中應該或不應該放什麼內容，而不會討論程式碼如何縮排等小議題。以一位成長中的程式設計人來說，當然要在所有程式內採用一致、容易閱讀的風格。

碁峰 www.gotop.com.tw

習題

本書含有許多程式設計的習題，但這本不是教科書，所以在書後也不會有習題的解答。這些習題提供了讀者能活用章節中所介紹概念的機會。當然，是否要解題都取決於您自己，但是把這些概念付諸實踐是很重要的學習過程。簡單地讀過這本書並不會成就什麼厲害的結果。請記住，本書不會直接告知讀者每種解決問題的方案要怎麼做。透過活用本書展示的技巧，您就能開發自己的能力，並發現應該要做什麼。此外，提高自信並邁向成功是本書的另一個主要目標。實際上，有種很好的方法可以知道讀者是否已充分了解某種特定問題領域，那就是當您能夠很有自信地解決該領域的問題之時。整體來說，程式設計習題實作應該是充滿樂趣的，雖然有時候您可能不太想解題，但是解開這些程式設計問題會是一個有意義的挑戰。

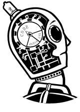
讀者應該把這本書當成大腦的障礙訓練場。障礙訓練場能增強力量、耐力和敏捷性，培養受訓者的信心。透過閱讀各章節的內容，並盡可能地把學到的概念活用在習題實作中，這樣可以建立自信，並培養出可用於任何程式設計情況的解決問題技巧。將來當您遇到難題時，就會知道應該怎麼搞定這類問題。

為什麼要用 C++？

書中的程式範例是用 C++ 所編寫的，話雖如此，但這本書是談解決程式問題，而不是專門針對 C++ 的。這裡不會有 C++ 的技巧和竅門之類的提示，而書中所教授的通用基礎概念可以在任何程式語言中使用。然而，要談程式設計就不能不講程式，因此還是需要選一種程式語言來代表。

會選用 C++ 的理由有很多種。首先，C++ 在各種問題領域都是主流使用的語言。其次，由於它源自於嚴謹的程序式 C 語言，所以 C++ 程式碼可以用程序式和物件導向模式來編寫。物件導向程式設計現在很普遍，所以在解決問題的討論中不能忽略它，但使用嚴謹的程序式程式術語來討論許多基本的解決問題概念，這樣可簡化程式碼和討論的內容。第三，作為一種具有高階程式庫的低階語言，C++ 允許我們運用兩個階層的程式設計。最好的程式設計師可以在需要時「手動連接」解決方案，並利用高階程式庫和 API 來減少開發時間。最後，選用 C++ 的一個原因是一旦學會了使用 C++ 解決問題，就能用其他任何程式語言來解決問題。很多程式設計師發現學了一種程式語言的技能後，可以輕鬆

碁峰 www.gotop.com.tw



地套用到其他程式語言中，對 C++ 來說更是如此，這是因為 C++ 的用法可跨域套用，但坦率地說，是由其難度來決定的。C++ 是真正用在實務上的，在程式設計時並沒有什麼輔助，剛開始時會讓人有些害怕，不過一旦開始在 C++ 中取得進展後，就會覺得自己不再是只會一點點程式設計的人了，而是即將成為一名真正的程式設計師。

碁峰
www.gotop.com.tw

第 1 章
解決問題的策略

本書所談的是關於問題的解決，但什麼是問題的解決（problem solving）呢？大家在平常對話中用到這個術語時，所表達的意涵與我們這裡所討論的含意完全不同。如果您的 1997 年份 Honda Civic 的排氣管冒出藍煙，怠速發動時有點震動，且油耗效率不佳，那麼可以利用專業汽車知識、診斷、更換零件，以及維修工具等來解決這個問題。如果您跟朋友講述這個問題，可能等到下列答案：「啊！您就把這輛車賣掉，換台新車，那問題就解決了」。但朋友的答案並不是問題真正的*解決方案*，這只是*逃避*問題的方法而已。

問題包含了限制，這個限制條件是問題不能違反的規則，或是解決問題的方式。以這台故障的 Civic 來看，其限制條件之一就是要修復這台車，而不是買台新車來解決。限制條件還可能包括維修的總成本、維修花費的時間，或者限制在維修時不能購買新的工具。

碁峰
www.gotop.com.tw

在解決程式問題時，也會遇到限制條件。常見的限制條件包括程式語言、平台（是在 PC、iPhone 或是其他平台上執行）、效能（遊戲程式可能要求每秒至少更新圖形 30 次，商業應用程式則可能依據使用者的輸入而有回應時間上限）或記憶體的需求量。有時候，限制條件還涉及引用其他程式碼，例如程式不能引入某些開放原始碼，或者相反，只能使用開放原始碼。

然後，對於程式設計師來說，我們可以把「**問題解決**」定義為編寫出原創的程式碼，用來執行特定工作，並滿足所有規定的限制條件。

新手程式設計師通常很渴望要完成定義中的第一部分，先編寫出程式來執行特定工作，但卻受挫失敗於定義的第二部分，也就是違反了規定的限制條件。我把這樣的程式稱為**小林丸號**（*Kobayashi Maru*），這程式看起來能產生正確的結果，但卻違反一個或多個限定的規則。如果讀者不熟悉小林丸號這個名字，那表示您對星艦迷航記 II：**星戰大怒吼**（*Star Trek II: The Wrath of Khan*）還不夠熟悉。這部電影中某個場景是星艦學院中充滿抱負的學員進行演習訓練的任務，其中學員被安置在模擬的艦橋上擔任船長，執行一項不可能的任務。無辜的人們會在受損的**小林丸號**戰艦上等待死亡，為了解救他們，需要與克林貢人開戰，但這場戰鬥只能以犧牲船長所在的戰艦告終。這場訓練的目的是為了測試學員在戰火中的勇氣。沒有致勝的方法，所有的選擇都會導致最壞的結果。在電影快要結束時，我們發現寇克船長作弊更改了模擬訓練的程式，讓演習是有贏的可能。寇克船長很聰明，但是他並沒有解決**小林丸號**的難題，只是避開了而已。

幸運的是，程式設計師所遇到的問題通常是可以解決的，但是許多程式設計師仍然使用寇克船長的方法。在某些情況下，他們是無意中做了這樣的選擇。（「哦！這個解決方案只有在資料項少於 100 個的時候才有效，但解決方案的處理量應該不受限才對，我必須重新思考一下。」）在某些情況下是故意取消限制條件，目的是為了應付老闆或老師要求的結案期限。還有些情況，程式設計師只是不知道怎麼滿足所有限制條件。在我所見過的最壞的例子中，程式設計課的學生花錢請人來幫忙寫程式。無論動機為何，我們都必須努力避免小林丸號的情況。

碁峰
www.gotop.com.tw

經典難題

在閱讀本書的過程中，您會注意到，就算原始程式碼的細節會因不同的問題領域而有差異，但是某些模式會一直出現在我們選用的方法中。這算是個好消息，因為無論是否對該問題領域有豐富的經驗，都能讓我們可以很有自信地解決各種問題。專家級的問題解決高手能夠快速識別**類比關係**，可辨識出已解決問題和未解決問題之間可利用的相似之處。如果我們辨識出問題 A 的特徵與問題 B 的特徵很相似，而我們已解決掉問題 B，那麼在解決問題 A 時就有了寶貴的洞察見解參考。

在本小節中，我們會討論程式設計領域以外的經典問題，把其中學到的教訓和經驗套用到程式設計的問題內。

狐狸、鵝與玉米

這裡要討論的第一個經典問題是關於需要渡河的農民所面對的難題。讀者以前也可能有遇過類似的難題。

問題：怎麼渡河？

有位農夫帶著狐狸、鵝與一袋玉米要渡河。農夫有艘船，但只能容納農夫自己和他的三個物品中的一個。不幸的是，狐狸與鵝都餓了。狐狸不能與鵝單獨在一起，否則狐狸會吃掉鵝。同樣地，鵝不能單獨與玉米在一起，不然鵝會吃掉玉米。農夫要怎麼樣才能把所有的物品都運過河呢？

這個問題的場景如圖 1-1 所示。如果您以前從未遇過這樣的問題，請停下來並花幾分鐘嘗試看看怎麼解開這個問題。如果您以前已解過這樣的問題，請試著回憶一下解決方案，看看是否能解決這個難題。

碁峯 www.gotop.com.tw

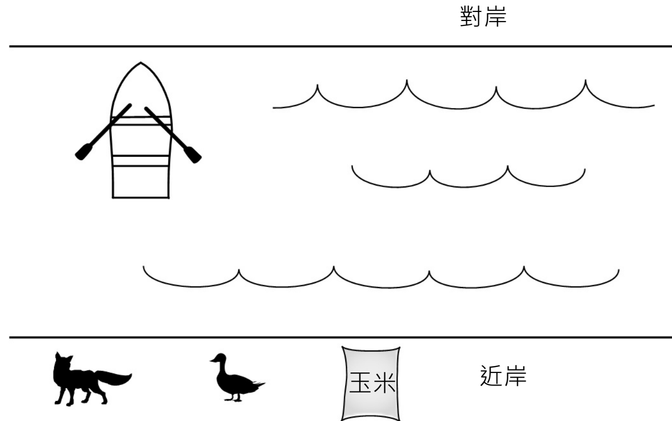


圖 1-1　狐狸、鵝與一袋玉米。船一次只能載一樣物品，狐狸不能單獨與鵝在一邊，鵝不能單獨與玉米在一邊

若沒有提示，很少人能解開這道難題，筆者也沒有這個能耐。以下是常見的思考推理方式。由於農夫一次只能帶一件物品，所以他需要多次來回才能把所有物品帶到對岸。在第一趟渡河的時候，如果農夫帶走狐狸，則鵝會單獨與玉米在一起，鵝會吃掉玉米。同樣地，如果農夫帶走玉米，則狐狸會單獨與鵝在一起，鵝會被狐狸吃掉。因此，農夫必須在第一趟渡河時帶走鵝，其結果如圖 1-2 所示。

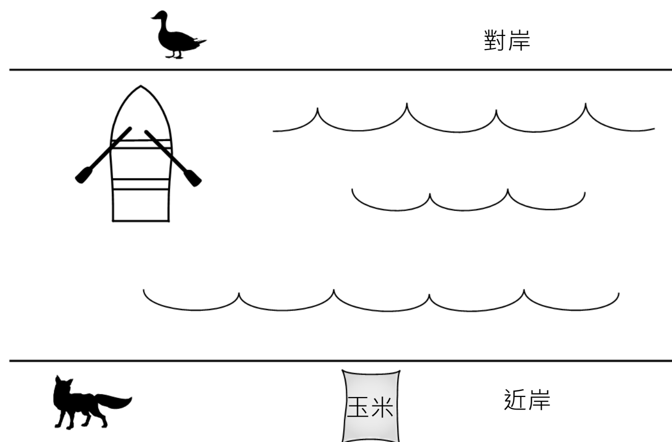


圖 1-2　解開狐狸、鵝與玉米難題所必須要採取的第一步。但是，從這步開始後，其他後續步驟似乎都以失敗告終

碁峰
www.gotop.com.tw

到目前為止，一切還好。但是在第二趟渡河中，農夫必須帶走狐狸或玉米。然而，無論農夫帶走了什麼，都必須與鵝放在對岸，而農夫要返回岸邊去拿剩下的物品。這表示狐狸與鵝或是鵝與玉米會一起放在對岸。由於這兩種情況都不能接受，因此這道難題看起來無解。

如果讀者以前有碰過這個問題，很可能還記得解題的關鍵元素。如前所述，農夫必須在第一趟渡河時接走鵝。在第二趟渡河時，假設農夫帶走狐狸，但農夫沒有把鵝留在對岸與狐狸在一起，而是**把鵝再帶回了**岸邊。然後，農夫把這袋玉米載過去，把玉米和狐狸留在對岸，最後第四趟再帶鵝過去。如圖 1-3 示範了完整的解決方案。

這道題目很難解，因為大多數人不會想要把其中一項物品從對岸帶回。甚至有人會說這個問題不公平，例如會說：「你又沒有說可以帶回物品！」。的確如此，但問題描述中也沒有明文禁止帶回物品呀。

想想看，如果明確指出可以把物品帶回來，那解開這道難題的難度就簡單多了：「農夫有艘船，**可來回載送物品**，但只能容納農夫自己和他的三個物品中的一個…」。有了這樣的提示建議，很多人都能發現問題所在。這說明了解決問題的一項重要原則：「如果您不知道所有可採取的動作（actions），則可能無法解決問題」。我們把這些動作稱之為操作（operations），藉由列舉出所有可能的操作就能解決很多問題，我們只要測試每種操作組合，直到找出可行的方案即可。一般來說，使用更形式化的術語來重述問題，通常能發現原本被忽略的解決方案。

讓我們忘記已知道的解決方案，並試著以更形式化的用語來描述這道難題。首先，我們列出限制條件，其關鍵的限制條件為：

1. 農夫在船上一次只能帶一件物品。

2. 狐狸不能與鵝單獨在同一岸邊。

3. 鵝不能與玉米單獨在同一岸邊。

碁峰 www.gotop.com.tw

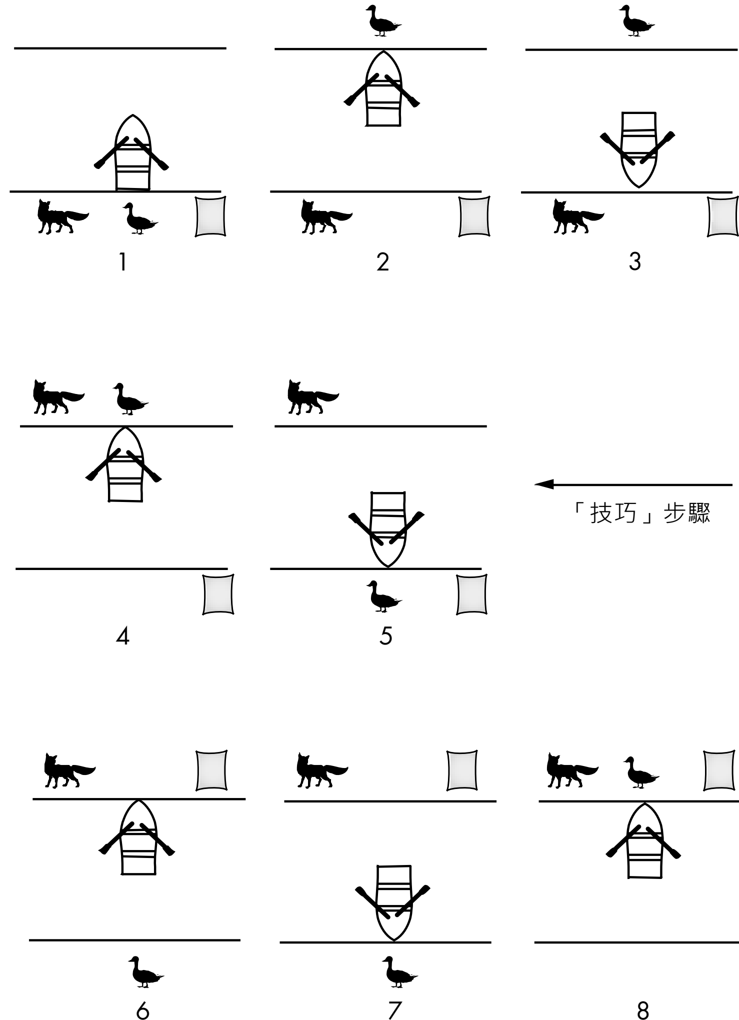


圖 1-3　狐狸、鵝與玉米難題的逐步解決方案

這道問題是說明限制條件重要性的好範例。如果把所有限制條件都去掉，那就很容易解開這道難題。假如去掉第一項限制條件，那麼很簡單地一趟載走 3 項物品渡河即可。就算一趟只載 2 件物品，也能讓狐狸和玉米先渡河，然後再帶鵝過去。假如去掉第二項限制條件（但保留另外 2 項限制），只需要小心先帶鵝渡河，再帶狐狸渡河，然後載玉米過去。因此，如果忘記或忽略了任何一項限制條件，就會碰到像小林丸號戰艦那樣的情況。

碁峯
www.gotop.com.tw

接下來是列出所有的操作。描述這道問題的操作有多種不同的方式，這裡列出我們認為可以採取行動的具體清單：

1. 操作：把狐狸帶到河的對岸。
2. 操作：把鵝帶到河的對岸。
3. 操作：把玉米帶到河的對岸。

請記住，以形式化來重述問題的目的是為了洞察發現問題的解決方案。除非已經解決了問題並發現了「隱藏的」可能的操作，也就是把鵝再帶回到岸邊，不然我們無法以上述動作清單來發現這項操作。因此，我們應該試著讓操作變得更通用或參數化。

1. 操作：把船從這一岸划到另一岸。
2. 操作：如果船空了，從岸上裝載一項物品。
3. 操作：如果船不為空，則把物品卸載到岸上。

藉由最通用的術語用詞來思考問題，上面的操作清單就能讓我們解決問題，而不會期待有「啊哈！可以把鵝帶回近岸」這樣憑空而生的靈感。如果我們列舉出所有可能的行動序列，並在違反限制條件之一或出現之前已看過的配置時，就終止這個序列，那麼最終會依照圖 1-3 的操作序列來解決這道難題。利用形式化重述問題的限制條件和操作，可避開問題原有的困難。

學到的經驗和教訓

我們從狐狸、鵝與玉米難題學到什麼呢？

以更形式化的方式重述問題是很好技術，可讓我們對問題有更深入的洞察。許多程式設計師會找其他程式設計師一起討論問題，這不僅是因為別的程式設計師可能會有答案，而是因為在與別人重述問題時常常會引發新的有用想法。重述問題就好像在與其他程式設計師一起討論，只不過您是同時扮演兩個角色。

更深遠的來看，思考問題與思考解決方案都具有生產力，在某些情況下甚至更具生產力。在許多情況下，找出解決方案的正確途徑往往本身就是解決方案。

碁峰 www.gotop.com.tw

滑塊拼圖遊戲

滑塊拼圖遊戲（sliding tile puzzle）有很多種不同的規格大小，如後面所見到的，這類遊戲提供某種特定的解決機制。以下是 3×3 版本的滑塊拼圖遊戲。

問題：八個數字推盤

一個 3×3 的格子中填入了八個滑塊（編號從 1 到 8）和一個空格。初始時格子中的滑塊是隨意混亂放置。滑塊可推入相鄰的空格，而滑塊之前的位置則變為空格。此問題的目標是重新滑動排放格子中的滑塊，讓它變成從左上角開始依數字有序地配置排放。

如圖 1-4 為這個問題所要完成的目標。如果讀者以前沒有玩過，可花點時間試試看。在網路上可找到很多滑塊拼圖模擬器，但為了學習和體會，最好使用紙牌或索引卡片製作出可以手動的桌遊。建議的起始排放配置如圖 1-5 所示。

1	2	3
4	5	6
7	8	

圖 1-4　這是滑塊拼圖中八數字版本的完成目標。空格表示相鄰滑塊可滑入

4	7	2
8	6	1
3	5	

圖 1-5　滑塊拼圖遊戲的某個起始排放配置

碁峰
www.gotop.com.tw

這個問題與前面的狐狸、鵝與玉米的問題完全不同。渡河問題的難度來自於可能會忽略其中某種操作。在這個問題上並不會發生那樣的情形。不管是什麼樣的排放配置，空格周圍相鄰最多有四個滑塊，這些滑塊都可以滑入空格內。這樣已充分列舉了所有可能的操作。

這個問題的難度來自於解決方案需要一連串的漫長操作。一系列滑動操作可能會把某些滑塊移到正確的目標位置，同時又會把其他滑塊移出正確位置，或者可能讓某些滑塊更接近其正確目標位置，但同時又把其他滑塊推到遠離目標的位置。因此，很難找出什麼特定的操作會朝著最終的目標邁進。由於無法衡量進度，所以很難制訂策略。許多嘗試滑塊拼圖的人只是簡單隨機地移動滑塊來求解，希望剛好碰到通往目標排放配置的路徑。

儘管如此，滑塊拼圖遊戲仍有一些策略可用。為了示範其中的一種方法，讓我們思考一個較小的矩形網格（不是正方形）範例。

問題：五個數字推盤

有一個 2×3 的格子填放了 5 個滑塊，其編號由 4 到 8，還有一個空格。一開始時格子中數字滑塊的排放配置是亂放的。滑塊可滑入相鄰的空格中，而滑塊推過去後的原本位置變成空格。這個問題的目標是讓格子中的數字滑塊排放由左上角有序地排放配置。

讀者可能留意到這裡的數字滑塊編號是由 4 到 8，而不是從 1 到 5，其原因很快會解釋說明。

儘管這與八個數字推盤是相同的基本問題，但僅用 5 個滑塊會容易很多。請試著完成如圖 1-6 中所示的排放配置。

碁峰 www.gotop.com.tw

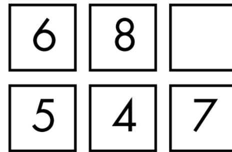


圖 1-6　2×3 縮小版的滑塊拼圖遊戲的某個特定起始排放配置

如果您只花了幾分鐘玩一下這些滑塊，很可能就會找到解決方案。藉由數字較少量的滑塊拼圖為例，可開發出特定的技巧。這項解題技巧再配合稍後進行的討論，就能用來解開所有的滑塊拼圖遊戲。

我稱這項技巧為看成一串「火車（*train*）」。根據觀察的結果，這會是包含空格的一串滑塊所形成的迴路，這些滑塊可沿著迴路的位置旋轉滑動，同時還保留滑塊的相對順序。圖 1-7 示範了四個滑塊所組成的最小可能火車。在第一種排放配置中，1 可以滑入空格，2 可滑入 1 所騰出的空格，最後 3 可滑入 2 所騰出的空格。這樣留下了與 1 相鄰的空格，使火車滑塊可以繼續滑動，因此，這些滑塊可以沿著火車路徑有效地旋轉到任意位置。

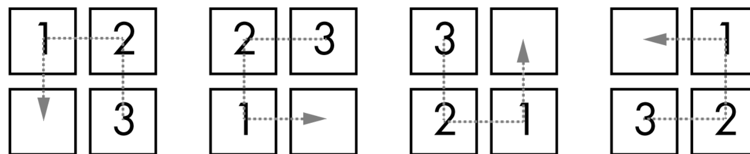


圖 1-7　這串「火車」指的是與空格相鄰的一串滑塊路徑，滑動時像一列火車串連行駛

我們可以火車串列的方式移動一系列滑塊，並保持其相對關係。現在回到前述 2×3 的格子排放配置。雖然這個格子配置中的所有數字滑塊均未放在正確的最終位置，但有些滑塊與在最終配置邊界的滑塊相鄰。舉例來說，在最終的排放配置中，數字 4 會位於 7 的上方，而且目前這些滑塊是相鄰的。如圖 1-8 所示，我們可以用六個位置的這串火車來滑動，把 4 和 7 滑到最終的正確位置。當我們這樣做的時候，剩下的滑塊幾乎都會到正確的位置上，最後僅需把數字 8 滑過去就完成了。

碁峰
www.gotop.com.tw

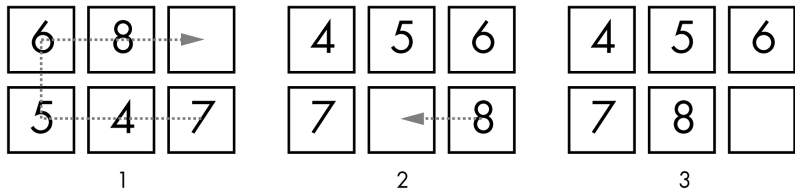


圖 1-8　從配置 1 開始，「火車」串列沿著路徑進行兩次滑動旋轉來到配置 2。隨後只要滑動一個滑塊就達成最終的配置 3

這樣的技巧怎樣讓我們解決各種滑塊拼圖難題呢？想一下原本的 3×3 配置例子。我們可以用六個位置的火車串列來滑動相鄰數字 1 和 2 的滑塊，直到數字 2 和 3 相鄰，如圖 1-9 所示。

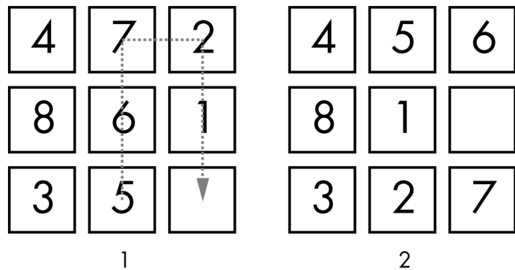


圖 1-9　從配置 1 開始，沿著「火車」路徑滑動旋轉到配置 2

這樣就會把 1、2 和 3 放置在相鄰的正方形中。接著使用八個位置的火車串列，這樣就能將 1、2 和 3 滑塊移到正確的最終位置，如圖 1-10 所示。

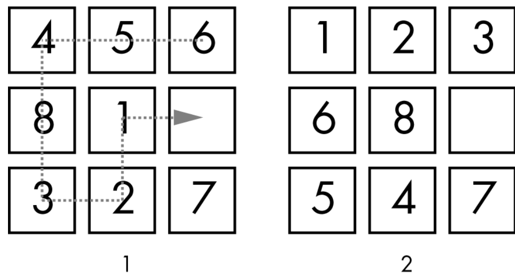


圖 1-10　從配置 1 開始，滑塊經過旋轉後移到配置 2，這時 1、2 和 3 滑塊就移到正確的最終位置

碁峯 www.gotop.com.tw

請留意滑塊 4-8 的位置。這些滑塊的配置正好和前面 2×3 格子配置的範例相同，這是個很關鍵的觀察。滑塊 1-3 已放在正確的位置，而剩下的滑塊可看成是更小型、更容易解決的獨立問題。請注意，要讓這樣的方法變得可行，就得要解開一整列或一整欄的滑塊。如果把滑塊 1 和 2 放在正確的位置，但滑塊 3 仍然不正確，則在不移動上一列中一個或兩個滑塊的情況下，是不可能把任何滑塊移到右上角的。

這相同的技巧可用來解決更大的滑塊拼圖難題。最大的常見滑塊拼圖是 15 塊，也就是 4×4 的格子。可先把數字 1-4 的滑塊移動到其正確位置，分解成 3×4 的格子大小，然後再滑動完成最左側欄的滑塊，變成 3×3 的格子，以此類推來解決這個問題。到這時，此問題已簡化為 8 塊的滑塊拼圖難題。

學到的經驗和教訓

我們從滑塊拼圖遊戲中學到什麼呢？

由於滑塊要移動的次數相當多，以致於很難也不可能從初始配置中，就為滑塊拼圖難題制訂出完整的解決方案。但是，無法規劃出完整的解決方案並不能阻止我們制訂策略或採用技巧來系統地解決難題。在解決程式設計問題時，有時候會遇到找不出解決方案清晰路徑的情況，但我們絕不會以此為藉口放棄規劃和系統化的方法。制訂策略比透過反覆試誤來解決問題更好。

我透過滑塊拼圖難題開發了「火車」技術。通常我會在程式設計中使用類似的技術。在面對繁重的問題時，我會試著把問題先縮小成簡化的版本來求解。這樣的實驗常會帶來有價值的見解。

另一個經驗和教訓是，有時分解問題的方式並不是很明確。因為移動滑塊不僅會影響其本身，還會影響其接下來可能的移動，所以大家會認為滑動滑塊拼圖必須一步步解決，而不是分階段來解決。值得花點時間去找出分解問題的方法。就算找不到清晰的分解方式，也能學到一些關於該問題的知識，可以幫助您解決該問題。在解決問題時，無論是否達成特定目標，在心中有個特定方向總是比隨機亂試要好得多。

碁峰
www.gotop.com.tw