寫作解答過橋問題程式的研究

邵海祥 (Herbert Shiu)
hcshiu@geocities.com

前言

作者有鑑於現在的程式員主要爲寫作「堆砌式」的程式寫作模式，而程式員於此類程式寫作的模式中，不能對其寫作程式的概念和能力有所改善，亦失去了寫作程式的樂趣。因此，作者於一九九九年二月上旬在新聞組[1]內刊出了一個過橋問題（詳見問題說明），希望籍此提高其他網友對寫作非「堆砌式」程式的興趣，和提供交流寫作程式心得的機會。

作者會在本文章內盡力講解如何以寫作程式來解決該問題，希望讀者能有所得著，並能將當中的概念運用於其他程式問題上。因爲本文主要爲講解寫作程式的概念，所以作者假設讀者對寫作程式有一定的認識。

作者曾任職電腦教育中心，當時其中一種教授的程式語言爲 LOGO，所以對 LOGO 有所認識，深知這種程式語言因爲簡單易學，普遍用以教導年幼的孩童，很多人亦因此認爲這種程式語言只是一件「童時玩意」，一種很幼稚，很無聊的程式語言，作者亦相信很多導師亦不知道它其實是「人工智能」範疇裡其中一種程式語言。所以作者在本文內會以 LOGO 爲講解之用，並在本文的附錄中列出一個完整的 LOGO 程式，希望讀者看過這個實質只得近百行的 LOGO 程式後，會對這種程式語言另眼相看。

作者並不是什麼寫作程式的專家，而本文中所記載的程式只是作者以自己的方式來寫作，並不一定是最好的解決辦法。如讀者對本文章內有任何意見，請不吝指教。

問題說明

在一個沒有燈光的晚上，四個人拿著唯一的一支燈要過橋，但橋只可負荷兩個人，並且在過橋的時候是須要拿著燈的。這四個人過橋所需時間分別爲 1、2、5 和 10 分鐘，如果兩個同時過橋，所需時間爲過橋兩個人當中較長者（例如需時分別爲 1 分鐘和 10 分鐘兩個人同時過橋，所需時間便爲 10 分鐘）。問題是，這四個人要怎樣過橋，才能在 17 分鐘內全到橋的另一邊？

分析

因爲橋只能負荷兩個人和只有一支燈，所以很直接地假設過橋的方式便是由每程由起點至終點是兩個人同行，到達後再由在終點那裡其中一位拿燈回到起點，

[1] news://news.starzine.com/starforum.comp.programming/

如此類推，每一次往返在終點便會多一個人，直至最後只有兩個人在起點用一程到達終點。

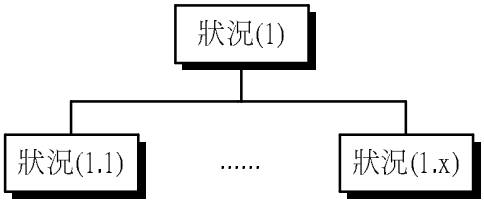
由此可知，如果須要過橋的人數爲 N，所需過橋次數便是 $2(N - 2) + 1$。所以對四個人來說，所需次數便爲五次。

電腦勝人腦？

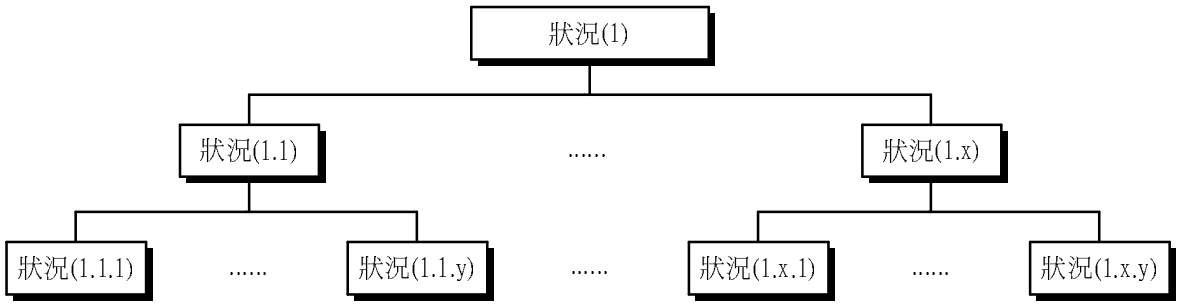
面對這類問題（另一個類似的問題如河內塔），之所以電腦比人腦優勝，並不是因爲它有比人更好腦袋，只不過是因爲它的運算速度和準確性高而已。另一方面，當人思考這類問題時，少不免有其主觀的想法，不自覺中走進了死胡同。所以要解決這個問題，便可以利用電腦高速運算的能耐來嘗試每個可行的方法。程式員的重任便只是以程式來教導電腦如何去逐一來嘗試，這亦是程式員面對最大的挑戰。

程式的概念

如果以問題一開始時爲首狀況，在這個狀況下是有很多可行的做法，每一個可行的做法的結果便成了另一個狀況，所以首狀況之下便有很多個可能的狀況。



跟著就著每個可能的狀況，亦有很多可行的做法，因此，在每個可能狀況下也有了很多可能的狀況。



如此類推，可以想像得到如果將所有可行的做法都列出來，這就變成了一棵以首狀況爲根的樹。如果這棵樹的其中一塊葉的狀況和問題所要求的結果一樣，那麼由根到這塊葉的路徑便是解決這個問題的方法了，同時，如果這棵樹有多過一塊葉的狀況等同要求的結果，意即這個問題有多於一個解答。否則，這個問題便是沒有解答了。

程式的設計

就著上一部份**程式的概念**裡所提及的概念，要解決這個問題，便要建立一個那裡所提出的一棵樹。當電腦建立了這棵樹後，便可以從根開始，用「深度優先[2]」的方法來「環遊」整棵樹，當巡查到樹的葉時，如果該葉的情況和問題所需的情況一樣，便即找到了答案，解答的方法便是由根到該葉的路徑了。

但在實際程式的寫作時，其實是不須要真真正正的去建立一棵完完整整的一棵樹的。一個很簡單的原因便是在建立這棵樹的時候，在每個情況時，便可一併檢查是否已到達了樹的葉（亦即是沒有其他的可行方法，針對本問題來說，到達了葉的時候便是當所有人已到達了目的地），而且，實際也沒有需要儲存已檢查了情況。所以在任何時間，程式只須儲存由首情況到正在檢查的情況之間的數據便可。

以本問題來實踐以上所提的方法是這樣的。本問題的首情況爲((1,2,5,10),())。（作者以這個方式來表達一個情況，第一個括號是所有在起點的人所需的過橋時間，而第二個括號是所有在目的地的人所需的過橋時間。）而在這個情況下，可行的方法便是所有從四個人中選出兩個人的方法，亦即是可以是(1,2)，(1,5)，(1,10)，(2,5)，(2,10)以及(5,10)。所以在首情況下是有六個可行的方法，程式處理了第一個可行的方法後便成了以下的狀況：

情況	時限	可行的方法
((1,2,5,10),())	17	**(1,2)** (1,5) (1,10) (2,5) (2,10) (5,10)
((5,10),(1,2))	15	(1) (2)

（以上以粗體表示的是在那個情況下所選取的方法，所以出現了下一個情況。）

跟著，在最後檢查的情況再選出一個可行的方法來處理：

情況	時限	可行的方法
((1,2,5,10),())	17	**(1,2)** (1,5) (1,10) (2,5) (2,10) (5,10)
((5,10),(1,2))	15	**(1)** (2)
((1,5,10),(2))	14	(1,5) (1,10) (5,10)

同樣地，再選出一個可行的方法來處理：

情況	時限	可行的方法
((1,2,5,10),())	17	**(1,2)** (1,5) (1,10) (2,5) (2,10) (5,10)
((5,10),(1,2))	15	**(1)** (2)

[2] Depth First Search

((1,5,10),(2))	14	**(1,5)** (1,10) (5,10)
((10),(1,2,5))	9	(1) (2) (5)

再繼續，選出一個可行的方法：

情況	時限	可行的方法
((1,2,5,10),())	17	**(1,2)** (1,5) (1,10) (2,5) (2,10) (5,10)
((5,10),(1,2))	15	**(1)** (2)
((1,5,10),(2))	14	**(1,5)** (1,10) (5,10)
((10),(1,2,5))	9	**(1)** (2) (5)
((1,10),(2,5))	8	(1,10)

因爲時限比可行的方法還少，所以在這情況下沒有任何可行的方法，亦即不用再測試了，而且現情況也永遠不須再使用，所以即使建爲樹的其中一葉或儲存起來也沒有任何用途，所以便可簡單地回復至之前一個情況，再試下一個可行的方法：

情況	時限	可行的方法
((1,2,5,10),())	17	**(1,2)** (1,5) (1,10) (2,5) (2,10) (5,10)
((5,10),(1,2))	15	**(1)** (2)
((1,5,10),(2))	14	**(1,5)** (1,10) (5,10)
((10),(1,2,5))	9	(1) **(2)** (5)
((2,10),(1,5))	7	(2,10)

同樣，現情況是沒有任何可行的方法，所以又回復至前一個情況再試下一個可行方法（即是(5)）：

情況	時限	可行的方法
((1,2,5,10),())	17	**(1,2)** (1,5) (1,10) (2,5) (2,10) (5,10)
((5,10),(1,2))	15	**(1)** (2)
((1,5,10),(2))	14	**(1,5)** (1,10) (5,10)
((10),(1,2,5))	9	(1) (2) **(5)**
((5,10),(1,2))	4	(5,10)

但同樣地也沒有可能的方法達到要求的結果，所以又回復至之前一個情況((10,(1,2,5))。這時，沒有任何可行的方法了，所以再回到之前的一個情況（(1,5,10),(2)），便選出下一個可行的方法來處理（(1,10)），亦即：

情況	時限	可行的方法
((1,2,5,10),())	17	**(1,2)** (1,5) (1,10) (2,5) (2,10) (5,10)
((5,10),(1,2))	15	**(1)** (2)
((1,5,10),(2))	14	(1,5) **(1,10)** (5,10)
((5),(1,2,10))	4	(1) (2) **(**10**)**

如此類推，便可以發現程式好像在進行「深度優先」的環遊，直至找到了答案或是已環遊了整棵樹了。所以便沒有需要真真正正的用記憶體來儲存一棵樹。需要儲存的，便只是在以上表內那些資料便足夠。

由程式的運作可以看出程式程式是需要一個「後入先出」的數據結構的，程式員可在程式內以程式來達到這個效果。但其實大部份高階程式語言也提供了一個更方便，更簡單的方法，便是迴遞(Recursion)。

另外一個不儲存整棵樹的原因是，對某些問題來說，整棵樹可能是很大的。就以本問題來說：

樹的層次	情況數量	附註
第一層	1	首情況
第二層	6	四人選出二人($_4C_2$)
第三層	6 × 2 = 12	在這六個情況下，在各自目的地二人選出一人
第四層	12 × 3 = 36	在起點三人選出二人($_3C_2$)
第五層	36 × 3 = 108	在目的地三人選出一人
第六層	108 × 1 = 108	在起點二人選出二人

由以上的表便可計算出，要儲存四人過橋問題的樹所需要的記憶體爲

1 + 6 + 12 + 36 + 108 + 108 = 271

即是 271 個情況所需的記憶體的數量。如果過橋的是五人，又或更多，所需的記憶體便會以幾何級數增加。

程式說明

本部份會根據之前所提出的方法，以 LOGO 來寫作對應的程式。

在 LOGO 程式語言中如果要儲存複雜的數據結構，可以使用其特有的列(List)來處理。使用列的好處是 LOGO 程式語言已經有很多處理列的函數。而且列的內容可以是一個基本的數據，又或是另一個列。因此，如果懂得運用 LOGO 這種特別的數據結構，便可以寫出很具彈性的程式。

在處理這個問題，列是用於儲存所有複雜的數據。例如在起點或目的地的人數是以一個儲存所有人所需時間的列來表示（例：[1 2 5 10]）；所有可行的方法是以一個如列爲內容的列來表示，例如[[1 2] [1 5] [1 10] [2 5] [2 10] [5 10]]。另外一個複雜的數據是已做了的方法，例如[[1 2] [1] [5 10] [2] [1 2]]，在這個列

裡，內容爲兩個數目爲列的表示由起點至目的地的人，而內容爲一個數目的表示由目的地回到起點。（爲要使這個列更容易令人明白，在建議程式中是有一個程式來格式化這個列的。）

以列來儲存數據另一個好處是它某個程度上和集(Set)很相似，所以在建議程式中處理數據，會使用了一些集的運算，例如集的差(Set Difference)和聯合(Set Union)等等。

針對這個問題，最主要的便是以下這個程式：

```
TO CrossBridge :Source :Target :Limit :Action :Direction
       IF :Limit < 0 [STOP]
       IF :Source = [] ~
       [      PRINT FormatSolution :Action :Limit
              STOP
       ]
       IFELSE :Direction = 1 ~
              [CrossTo :Source :Target :Limit :Action GetTwoItems :Source] ~
              [CrossBack :Source :Target :Limit :Action GetItem :Target]
END
```

此程式所要求的輸入資料是，所有起點的人過橋所需時間(:Source)，所有目的地的人過橋所需時間(:Target)，所餘時限(:Limit)，由首情況到現情況所用的方法(:Action)，過橋方向(:Direction，1 代表由起點到目的地，2 則代表由目的地回到起點)。

首先檢查是否已過了時限，如果已起過了時間，便沒有需要再試什麼。否則，檢查是否起點已沒有人，如果是這樣，便可以列出答案。否則，便根據過橋的方向呼叫對應的程式。請留意由起點到目的地是呼叫程式 CrossTo，而最後一個輸出至 CrossTo 的資料是用 GetTwoItems[3] :Source 來取得，意即是將所有可以在起點選出兩個人的方法輸出至 CrossTo 程式。同樣地，由目的地回到起點，最後一個資料是用 GetItem[4] :Target 來取得，意即是將所有在目的地可以選出一個人的方法輸出至 CrossBack 程式。

呼叫這個程式時便要輸入問題開始時的情況，如：

[3] 如果呼叫 GetTwoItems 並以[1 2 5 10]爲輸入，輸出爲[[1 2] [1 5] [1 10] [2 5] [2 10] [5 10]]

[4] 如果呼叫 GetItem 並以[1 2 5 10]爲輸入，輸出爲[[1] [2] [5] [10]]。這個程式看似多餘，但其目的是和 GetTwoItems 對應，令 CrossTo 和 CrossBack 的內容更相近。

資料名稱	內容
:Source	[1 2 5 10]
:Target	[]
:Limit	17
:Action	[]
:Direction	1

以下便是 CrossTo 程式的內容：

```
TO CrossTo :Source :Target :Limit :Action :BeingTest
       IF :BeingTest = [] [STOP]
       CrossBridge ~
              SetDifference ~
                     :Source ~
                     FIRST :BeingTest ~
              Union ~
                     :Target ~
                     FIRST :BeingTest ~
              :Limit - Max ~
                     FIRST FIRST :BeingTest ~
                     FIRST BUTFIRST FIRST :BeingTest ~
              LPUT ~
                     FIRST :BeingTest ~
                     :Action ~
              -1
       CrossTo :Source :Target :Limit :Action BUTFIRST :BeingTest
END
```

因爲 CrossTo 首次是由 CrossBridge 來呼叫的，所以在最後一個輸入資料是所有可行的方法。

這個程式首先檢查如果是沒有可試的可行方法，便不須要做什麼。否則便針對第一個可試的方法，準備好資料來呼叫程式 CrossBridge 來繼續試在現況下所有可行的方法，例如起點的人會除去過橋的人，而目的地則增加了過橋的人，時限會減去過橋兩個人當中過橋需時的較多的時間，而已做了的步驟則加進過橋兩個人的時間，而方向則是由目的地返回起點的。試完在現情況下所有可行的方法之後，便再測試在之前情況其他可行的方法。

由目的地回到起點的程式 CrossBack 和 CrossTo 很相似，其內容如下：

```
TO CrossBack :Source :Target :Limit :Action :BeingTest
       IF :BeingTest = [] [STOP]
       CrossBridge ~
              Union ~
                     :Source ~
                     FIRST :BeingTest ~
              SetDifference ~
                     :Target ~
```

```
                FIRST :BeingTest ~
            :Limit - FIRST FIRST :BeingTest ~
            LPUT ~
                FIRST :BeingTest ~
                :Action ~
            1
      CrossBack :Source :Target :Limit :Action BUTFIRST :BeingTest
END
```

這個程式和 CrossTo 的結構是相同的，所以不再詳加講解了。

整個解決過橋問題便主要是由這三個程式所組成，其他的程式是提供處理集(Set)的運作，如 Union，SetDifference，和找出兩個數之中較大的一個數值等等。附錄已列出完整的程式，讀者請參考該程式來更進一步明白整個程式是如何運作。

結語

本文內的建議程式是以 LOGO 來寫的，但其實除了它使用了 LOGO 特有的列外，它的結構和其他高階程式語言所寫的程式其實沒多大分別。所以讀者如果有興趣使用其慣於使用的程式語言來寫同樣的程式，只須在程式內自行建立類似列的數據結構，（尤以物件導向程式語言[5]較爲合適），便可以以類似的程式結構來解決這個問題。

解決過橋問題的程式當然不只作者所提供的建議程式，讀者可以以對這個問題的認識來寫作程式來解決。希望讀者看完本文章後有所得著和啓發。

版權聲明

[5] Object Oriented Programming Langauge

附錄：建議程式

```
;-------------------------------------------------------------------------------
; Program Name : Bridge.logo
; Written By : Herbert Shiu
; Language : LOGO
; Interpreter : UCBLogo
; Output :
;      This program is used to solve the crossing bridge problem.
;      (See Problem Description)
; Problem Description :
;      Four men have to cross a bridge with 17 minutes with following
;      Conditions:
;      a) The bridge can support at most 2 men at a time.
;      b) Crossing a bridge requires a lamp and there is only one lamp.
;      c) The times used to cross the bridge are 1, 2, 5, 10 minutes
;         respectively.
;      d) If two men are crossing the bridge, the time required is the
;         longer one.
; Implications :
;      a) From the above description, the lamp must be passed to and back the
;         bridge.
;      b) Two men cross the bridge from source to target and one men cross
;         the bridge back to the source for bringing the lamp back.
; Usage :
;      Enter "go" at the prompt. And a sample input to the programs is:
;      First : "1 2 5 10" as time list
;      Second : "17" as time limit
; Modification History :
;      09-Jan-1999    Written.
;-------------------------------------------------------------------------------
; Copyright 1999 Herbert Shiu. All right reserved.
;-------------------------------------------------------------------------------

TO CrossBridge :Source :Target :Limit :Action :Direction
; Output : Prepare data for next step

; If no time left, just quit
       IF :Limit < 0 [STOP]
; If source become empty, i.e., successful, print the solution
       IF :Source = [] ~
       [      PRINT FormatSolution :Action :Limit
              STOP
       ]
; Prepare arguments according to the direction of next step
       IFELSE :Direction = 1 ~
              [CrossTo :Source :Target :Limit :Action GetTwoItems :Source] ~
              [CrossBack :Source :Target :Limit :Action GetItem :Target]
END

TO CrossBack :Source :Target :Limit :Action :BeingTest
; Output : Prepare the environment according to the current going back action.
;         for next steps. If there are other possible actions, try them.

; If no available action, just quit
       IF :BeingTest = [] [STOP]
```

```
; Set environment according to the current going back action
       CrossBridge ~
              Union ~
                     :Source ~
                     FIRST :BeingTest ~
              SetDifference ~
                     :Target ~
                     FIRST :BeingTest ~
              :Limit - FIRST FIRST :BeingTest ~
              LPUT ~
                     FIRST :BeingTest ~
                     :Action ~
              1
; Try again with another action
       CrossBack :Source :Target :Limit :Action BUTFIRST :BeingTest
END

TO CrossTo :Source :Target :Limit :Action :BeingTest
; Output : Prepare the environment according to the current going to action.
;         for next steps. If there are other possible actions, try them.

; If no available way, just quit
       IF :BeingTest = [] [STOP]
; Prepare the environment according to the current going to action
       CrossBridge ~
              SetDifference ~
                     :Source ~
                     FIRST :BeingTest ~
              Union ~
                     :Target ~
                     FIRST :BeingTest ~
              :Limit - Max ~
                     FIRST FIRST :BeingTest ~
                     FIRST BUTFIRST FIRST :BeingTest ~
              LPUT ~
                     FIRST :BeingTest ~
                     :Action ~
              -1
; Try again with another action
       CrossTo :Source :Target :Limit :Action BUTFIRST :BeingTest
END

TO FormatSolution :Solution :TimeLeft
; Output : Format the solution to be displayed

; If the end of solution is reached, just return the time left.
       IF :Solution = [] [OUTPUT SENTENCE ": :TimeLeft]
; Format the current action accordingly
       IFELSE (COUNT FIRST :Solution) = 2 ~
              [OUTPUT FPUT ~
                     SENTENCE "TO: FIRST :Solution ~
                     FormatSolution BUTFIRST :Solution :TimeLeft] ~
              [OUTPUT FPUT ~
                     SENTENCE "BACK: FIRST :Solution ~
                     FormatSolution BUTFIRST :Solution :TimeLeft]
END

TO GetItem :InputList
```

```
; Output : Generate a list with elements are lists of one element in the
;        input list.
       IF :InputList = [] [OUTPUT []]
       OUTPUT FPUT ~
              FPUT FIRST :InputList [] ~
              GetItem BUTFIRST :InputList
END

To GetTwoItems :InputList
; Output : Generate a list with all possible combinations of two items from
;        the input list.
       IFELSE :InputList = [] ~
              [OUTPUT []] ~
              [OUTPUT ~
                     Union ~
                            GenPairs FIRST :InputList BUTFIRST :InputList ~
                            GetTwoItems BUTFIRST :InputList]
END

TO Go
; Output : Get input from user and prepare arguments for calling the
;        problem solving predicate.
       (LOCAL "TimeList "TimeLimit)
       PRINT [Enter the list of time required:]
       MAKE "TimeList READLIST
       PRINT [Enter the total time limit:]
       MAKE "TimeLimit FIRST READLIST
       CrossBridge :TimeList [] :TimeLimit [] 1
END

TO Max :A :B
; Output : Output the maximum of two items
       IFELSE :A > :B ~
              [OUTPUT :A] ~
              [OUTPUT :B]
END

TO GenPairs :Item :InputList
; Output : Return a list which pair up each item in input list with the
;        input item.
       IFELSE :InputList = [] ~
              [OUTPUT []] ~
              [OUTPUT ~
                     FPUT ~
                            LIST :Item FIRST :InputList ~
                            GenPairs :Item BUTFIRST :InputList]
END

TO SetDifference :A :B
; Output : Return the set difference of A by B
       IF :A = [] ~
              [OUTPUT []]
       IFELSE MEMBERP FIRST :A :B ~
              [OUTPUT SetDifference BUTFIRST :A :B] ~
              [OUTPUT FPUT FIRST :A SetDifference BUTFIRST :A :B]
END

TO Union :A :B
```

```
; Output : Return the union of sets A and B
       IF :B = [] ~
              [OUTPUT :A]
       IFELSE MEMBERP LAST :B :A ~
              [OUTPUT Union :A BUTLAST :B] ~
              [OUTPUT LPUT LAST :B Union :A BUTLAST :B]
END

;---------------------------------------------------------------------------
; End of file
;---------------------------------------------------------------------------
```