以搜尋爲基礎的電腦化問題策略解決

國立臺南大學 黃國禎
數位學習科技系 教授
資訊教育研究所 所長
理工學院 院長

問題定義 (Problem formulation)

- 在問始運用電腦解決一個問題之前，首先要能將問題清楚的在電腦中描述。
- 完整的問題描通常包含大量與問題特性相關的資訊，而與這些資訊相關的知識對於找到解答有很大的幫助。

問題與解題狀態的定義(Well-defined problems and solutions)

- 利用狀態空間來定義問題有兩個基本要素:

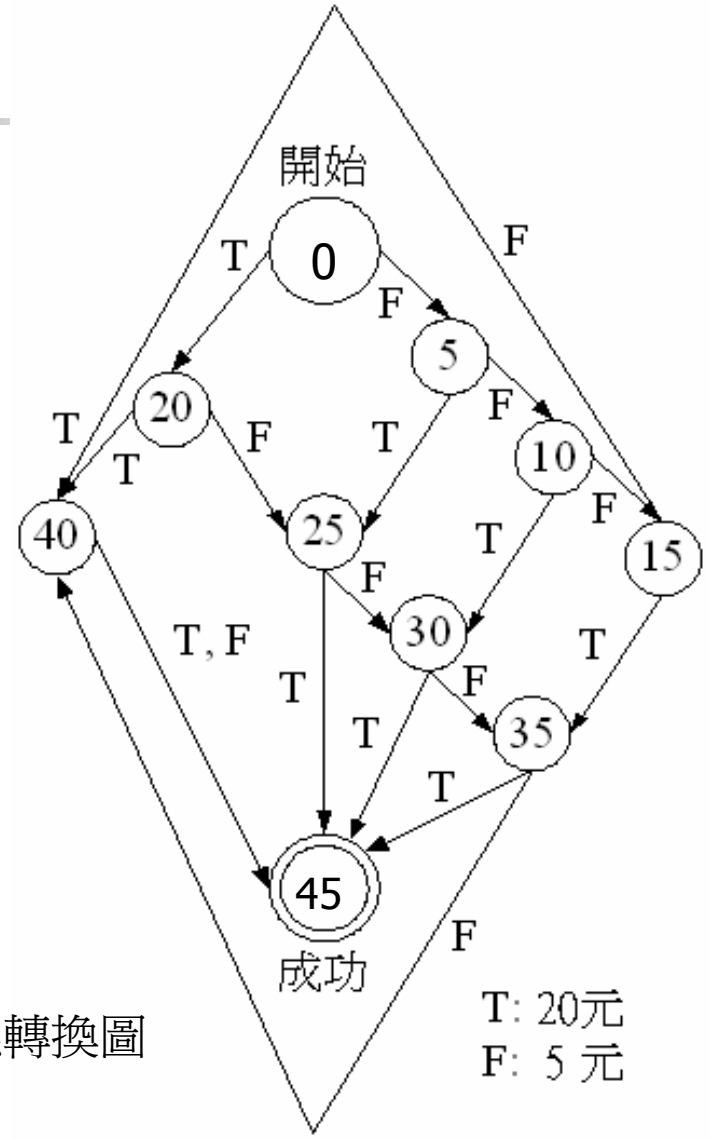
1. 狀態(states)
2. 行動(actions)

問題與解題狀態的定義(Well-defined problems and solutions)

- 具體的表達方式如下：
- 初始狀態(initial state)。
- 可用的運算元(operator)的集合。
- 「狀態空間」(state space)
- 「路徑」(path)：
- 「目標測試」(goal test)
- 「路徑成本」(path cost)

問題與解題狀態的定義(Well-defined problems and solutions)

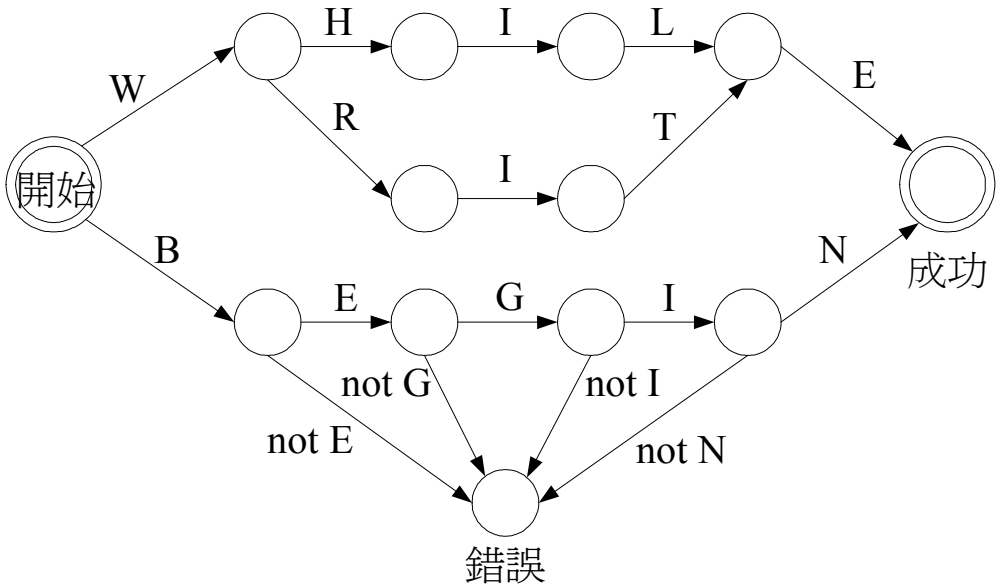
- 此圖稱爲「有限狀態機器」（Finite state machine）圖示法，因爲這個機器的狀態個數是有限的。
- 有限狀態機器常用於程式語言編譯器或其他電腦程式中，用來辨識輸入字字串的正確性。



<圖2.1 >牛肉乾販賣機狀態轉換圖

解題狀態的評量 (Measuring problem-solving performance)

- 此圖爲部分的有限狀態機器，用來測試輸入的字串是否正確。



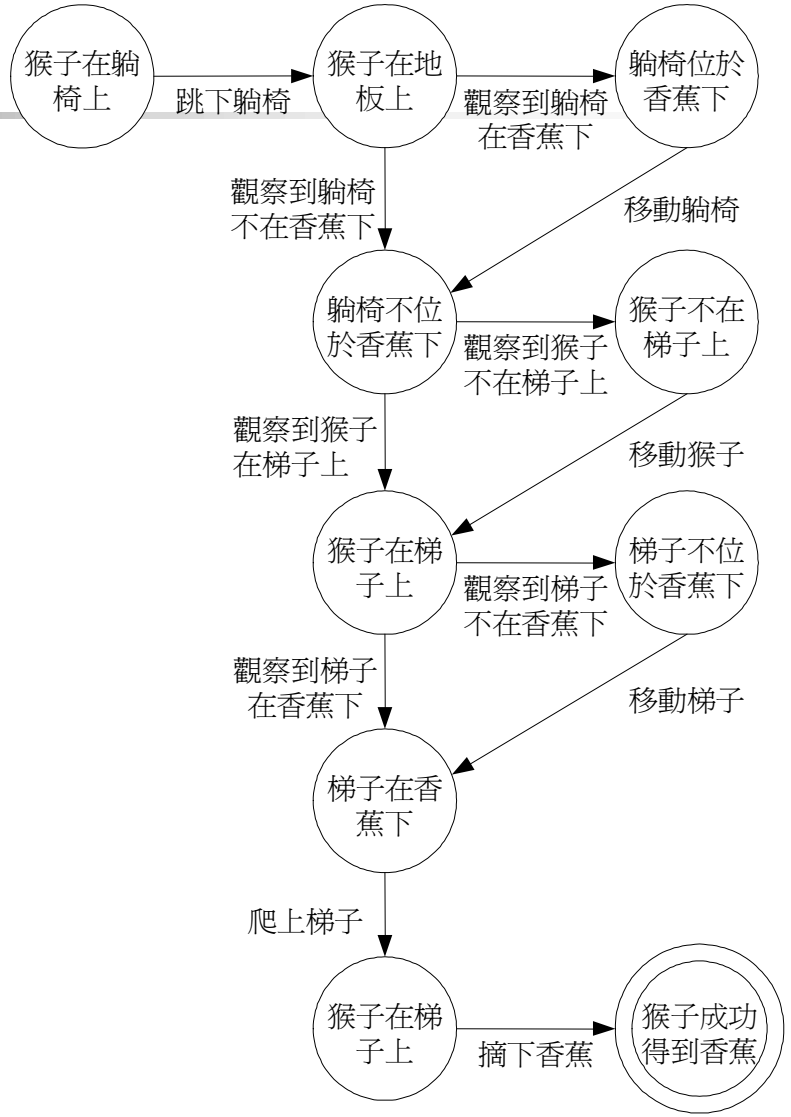
<圖2.2> 用以辨識WHILE、WRITE和BEGIN的有限狀態機器

解題狀態的評量 (Measuring problem-solving performance)

- 我們可以把狀態空間想像成問題空間(Problem space)，有些狀態是符合解決問題的中間部分，另一些狀態則是符合答案的。
- 以下的猴子與香蕉問題可看成是一個問題空間，只需要回應問題的“是”與“不是”，就能決定下一個狀態。

解題狀態的評量

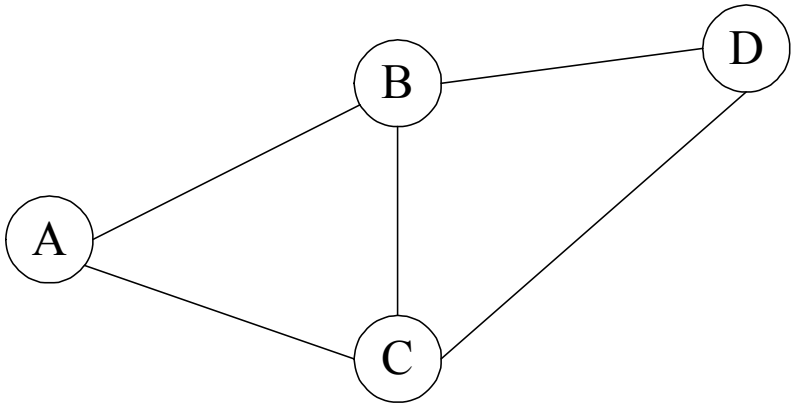
- 跳下躺椅
- 移動梯子
- 把梯子移到香蕉下的位置
- 爬上梯子
- 摘下香蕉



<圖2.3> 猴子拿香蕉的問題空間

解題狀態的評量 (Measuring problem-solving performance)

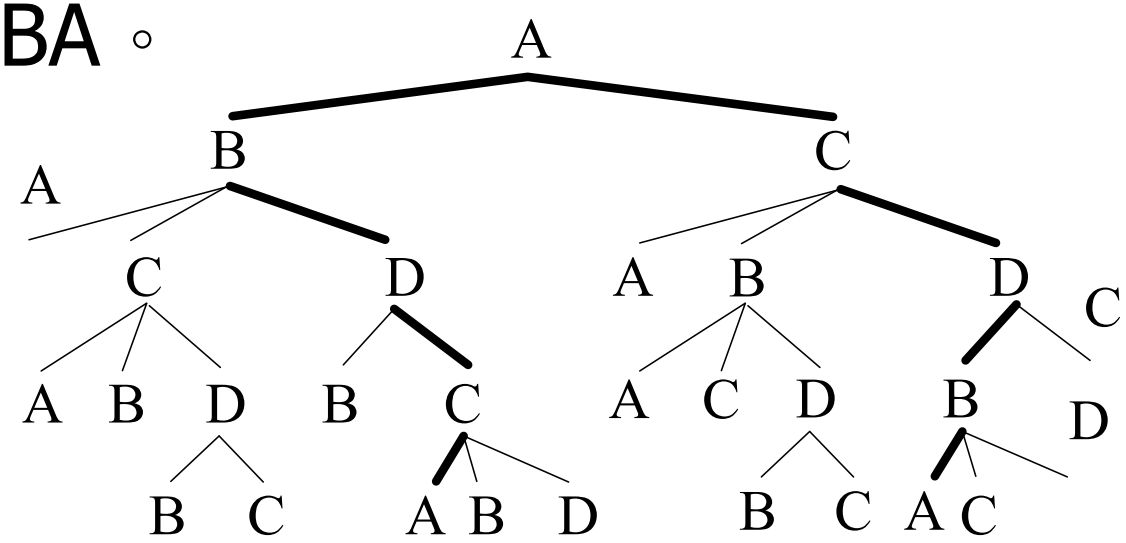
- 另一個圖形的應用是以探勘路徑找出問題的解法，圖 (a)表現了旅行推銷員(traveling Salesman)問題。



(a) 旅行推銷員的問題描述

解題狀態的評量 (Measuring problem-solving performance)

- 圖 (b)表示了從A點出發的樹，其中正確的路徑以粗線表示，如ABDCA或ACDBA。



(b) 搜尋路徑(粗線是解答路徑)

解題狀態的評量 (Measuring problem-solving performance)

- 一個狀態空間對於描繪結構不明確的問題仍相當有用
- 結構不明確的問題是指很多不確定因素的結合，這種不確定性可以用問題空間精確表達。

解題狀態的評量 (Measuring problem-solving performance)

- 表2.1是可能的不確定因素及旅遊代理人必須解決的問題。

特徵	客戶的反應
目標不明顯	我在想到底要去哪裡
問題空間範圍未被介定	我不確定要去哪裡
問題狀態不是離散的	我只是想去旅遊，目的地並不重要
中間的狀態不易實行	我沒有足夠的錢去
狀態的可用運算元未知	我不知道怎麼可以籌到錢
時間限制	我必須儘快出發

<表2.1> 旅遊非結構化問題的例子

解題狀態的評量 (Measuring problem-solving performance)

- 問題空間讓我們更明確地了解一個結構不明確的問題所具備的特徵，藉由描繪這些特徵的參數以找到合適的解答。
- 在旅遊問題中，狀態的範圍並未被界定，因爲旅行者會有無限可能的目的地。

解題狀態的評量 (Measuring problem-solving performance)

- 如果我們將每個讀數視爲一個狀態，那麼將會有無限個狀態，因爲每個狀態都是一個實數，而兩個實數間有無限個實數。
- 相反的，數位的計量器讀數的範圍是被清楚界定的，因爲它的狀態是離散的。

解題狀態的評量 (Measuring problem-solving performance)

- 搜尋的效能至少可以用三種方法來評量：

  (1)它是否能找到答案？

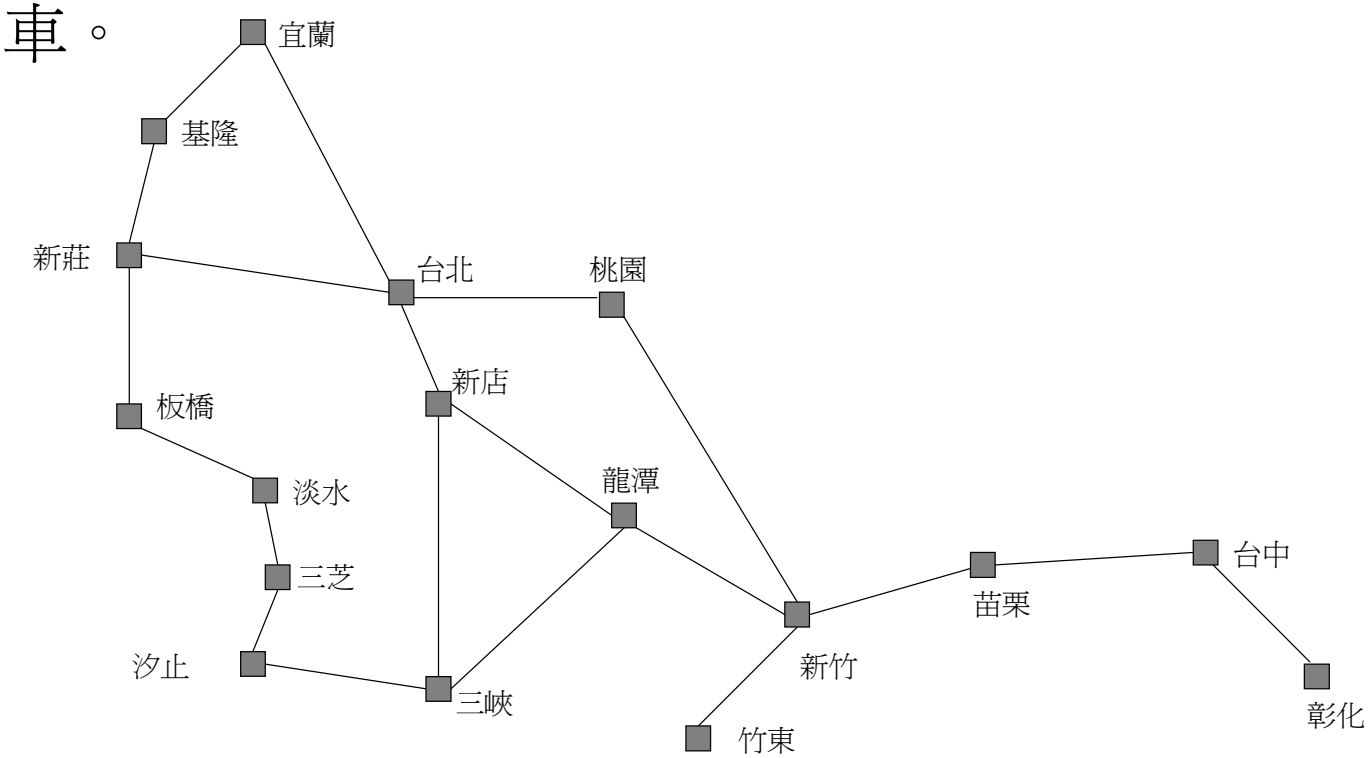
  (2)它是否爲好的答案？

  (3)需要找到解答的時間和空間的搜尋成本是什麼？

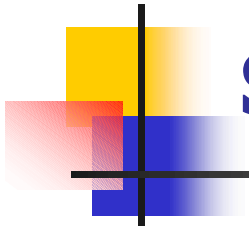
- 搜尋的總成本即是路徑成本和搜尋成本的總和。

狀態的選擇與執行 (Choosing states and actions)

- 初始狀態是「在新莊」而目標測試則是「是否在新竹？」運作元則是對應於沿著兩個城市之間的路駕車。



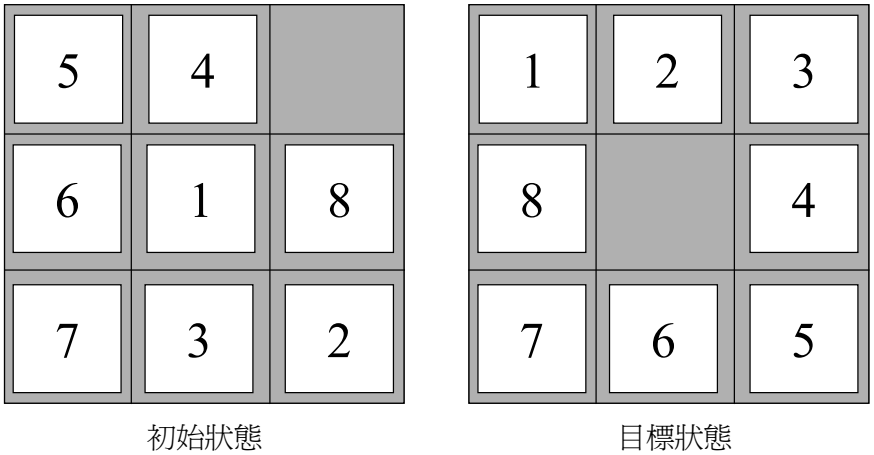
狀態的選擇與執行 (Choosing states and actions)



- 問題解決真正重要的，是決定要用什麼來描述狀態和動作，而什麼是可省略的。
- 這種從描述中省略細節的過程稱爲**抽象化(abstraction)**。

解題範例-以智慧盤遊戲爲例 (the 8-puzzle problem)

- 圖2.6是一3 x 3的智慧盤，其中有8個標示數字的方塊，以及一個空格。
- 與空格相鄰的方塊可以滑動到空格裡。



<圖2.6：3 x 3智慧盤>

解題範例-以智慧盤遊戲爲例 (the 8-puzzle problem)

- 一個重要的技巧是必須注意到，與其用「把四號方塊移動到空格裡」這個操作，不如用「空格和它左邊的方塊交換位置」的操作較爲敏銳。這麼一來，會導致下列公式：
  - 狀態：狀態描述要能分辨八個方塊個別在九個方格中的位置。爲了效率起見，最好也包含空格的位置。
  - 操作：空格向左、右、上、下移動。
  - 目標測試：狀態是否符合圖2.6所示的目標狀態。
  - 路徑成本：每一步成本是1，所以路徑成本就是路徑長度。

解題範例-以智慧盤遊戲爲例 (the 8-puzzle problem)

- 智慧盤屬於排列組合系列的問題，這種問題通常是NP-complete，亦即當問題中的方格個數增加時，解決問題需要的時間通常會以指數的次方成長。
- 其中15個方格的智慧盤問題常在人工智慧中被作爲測試新的搜尋演算法的標準問題。

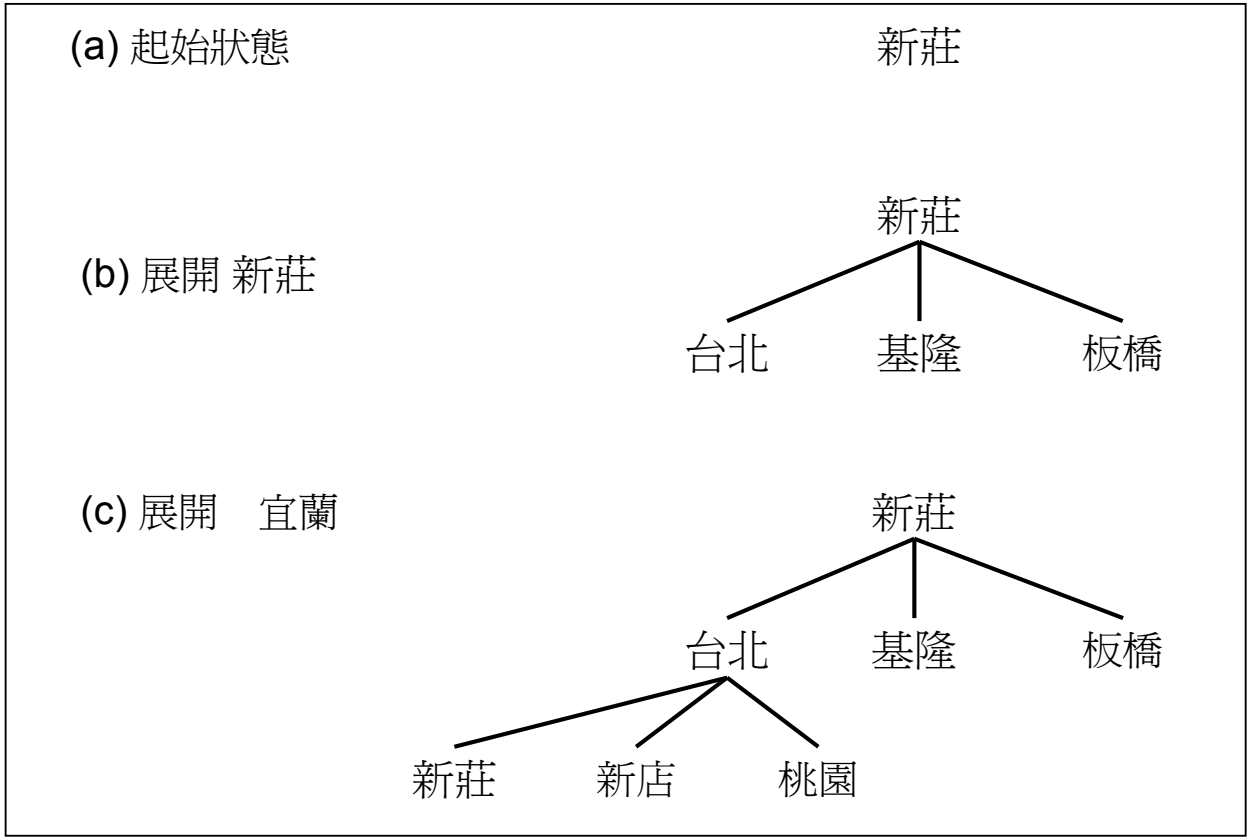
搜尋策略 (Searching Strategies)

- 對搜尋演算法的最簡單的詮釋方式，即是以狀態空間中搜尋樹的建構過程來進行說明。
- 其中由樹根直接展開的節點，標記爲深度是1；由深度是*d*的節點直接展開的節點，標記爲深度是*d*+1。

通用搜尋演算法 (General search algorithm)

- 圖2.7顯示通用搜尋演算法在建立從新莊到新竹搜尋樹的展開路線。
- 區別出狀態空間和搜尋樹是很重要的。在尋找城市路徑的問題上，雖然狀態空間上僅有17個狀態，但卻有無限多的可能路徑，因此在搜尋樹上會產生無限多的節點。

通用搜尋演算法 (General search algorithm)



<圖2.7> 從新莊到新竹搜尋樹的展開路線

通用搜尋演算法 (General search algorithm)

- 通用搜尋演算法：
- GENERAL-SEARCH (*問題描述*, *策略*) 傳回 解答 或 宣告失敗

  依初始狀態建立起始節點

  重複以下的指令

  IF 沒有可展開的節點 **THEN**傳回 宣告失敗

  依據傳入的*策略*決定展開的節點

  IF 有任何節點爲*目標節點***then**傳回 解答

  ELSE 將展開後的節點加入搜尋樹中

先寬後深搜尋策略 (Breadth-first search)

- 在這個策略中，根節點會先被展開，然後再展開被根節點展開的所有節點，接著再展開它們的後繼者等。

- 在寬度優先的搜尋樹中，所有在深度爲d的節點皆會比所有在深度爲d+1的節點早被展開。

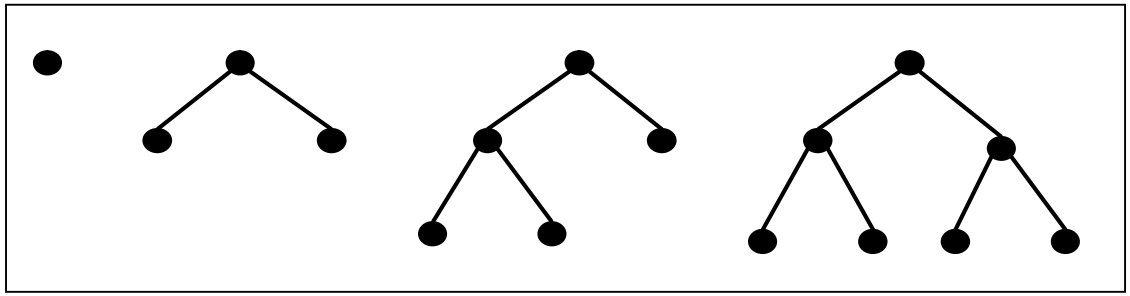
先寬後深搜尋策略 (Breadth-first search)

- 寬度優先搜尋法可以用先前介紹的通用搜尋演算法來開發，只要運用一個佇列函數(Queuing function)將新展開的狀態放入佇列的最後面：

function BreadthFirst-Search (*問題描述*) 傳回 解答 或 宣告失敗
  傳回 GENERAL-SEARCH (*問題描述*, 新展開的狀態放入佇列的最後面)

先寬後深搜尋策略 (Breadth-first search)

- 寬度優先搜尋法是一個非常系統化的策略，因爲它會先考慮所有長度爲1的路徑，然後是那些長度爲2的路徑…等。
- 如果有解答，寬度優先搜尋法會很快的發現它；
- 而如果有多個解答，寬度優先搜尋法將會先發現最小路徑的目標。



<圖2.8> 寬度優先搜尋樹的範例

先深後寬搜尋策略 (Depth-first search)

- 深度優先搜尋法總是先展開一棵搜尋樹中位於最深一層的節點，只有當遇到末端節點時，才會回溯到上一層，並展開另一個較淺層的節點。

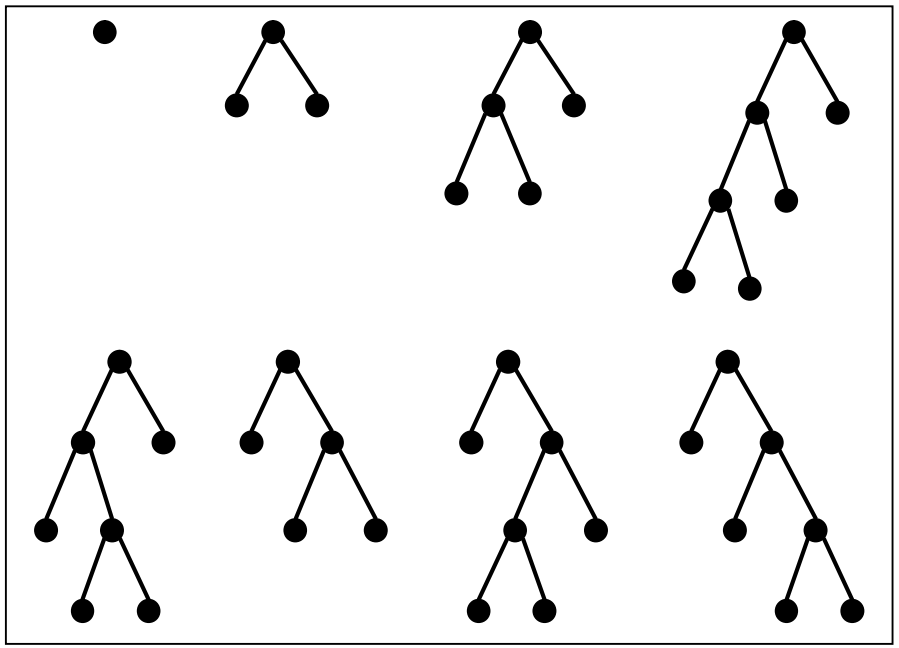
先深後寬搜尋策略 (Depth-first search)

- 深度優先搜尋法在實作時可應用通用搜尋法，將新展開的節點放在佇列的前面：

function DEPTH-FIRST-SEARCH (*問題描述*) 傳回 解答 或 宣告失敗
  傳回GENERAL-SEARCH (*問題描述*, 新展開的狀態放入佇列的最前面)

先深後寬搜尋策略 (Depth-first search)

- 由於新展開的節點是最深的，它的後繼者將是更深的節點，如圖2.9所示。



<圖2.9> 深度優先搜尋樹範例

先深後寬搜尋策略 (Depth-first search)

- 深度優先搜尋法的缺點爲，它可能會陷在錯誤的路徑中。

- 因此深度優先搜尋法可能會無法從錯誤的選擇路徑中恢復到靠近搜尋樹頂端的節點

先深後寬搜尋策略 (Depth-first search)

- 即使有較淺層的解答存在時，搜尋將可能持續向更深層的方向展開。

- 若無適當的控制，深度優先搜尋法將陷在無窮迴圈，即使最終會找到解答路徑，而這些路徑可能比最佳路徑長。

最佳解優先搜尋策略(best-first search)

- 這個搜尋法只是根據最佳化的評估函數來選擇下一個搜尋的節點。
- 當評估函數的準確度愈高，則愈可能找到最佳的節點
- 反之，評估函數可能無作用，甚至可能導至錯誤的搜尋。

最佳解優先搜尋策略(best-first search)

- 它使用一些估計解答成本的測量方式，並設法朝最低成本的方向進行搜尋。
- 了使搜尋更精確，因此估計解答成本的測量必須加上到達結束狀態的路徑成本估計。

最佳解優先搜尋策略(best-first search)

- 以下爲使用通用搜尋演算法來描述「最佳解優先搜尋法」的方式：

function BEST-FIRST-SEARCH (*問題描述*, 評估函數) 傳回 解答 或 宣告失敗
順序函數 ← 依評估函數決定的節點展開順序
傳回GENERAL-SEARCH (*問題描述*, 順序函數)

最佳解優先搜尋策略(best-first search)

- 以井字遊戲爲例，若電腦是“ｏ”，對手是“x”，且評估函數定義爲：
- F(x)=同一行及對角線的其他空白個數×1
  +同一行及對角線的“ｏ”個數×2+ Y(X)
- Y(X)= 0 若該一行及對角線的“x”個數爲0
  = –1 若該一行及對角線的“x”個數爲1
  = 4 若該一行及對角線的“x”個數爲2

最佳解優先搜尋策略(best-first search)

- 假設目前的盤面狀況如下：
- 一開始全部都是空格

A1	A2	A3
A4	A5	A6
A7	A8	A9

最佳解優先搜尋策略(best-first search)

若由電腦先下，則

- F(A1)=(2+2+2) ×1 + (0+0+0) × 2 + (0+0+0) = 6
- F(A2)=(2+2) ×1 + (0+0) × 2 + (0+0) = 4
- F(A3)=(2+2+2) ×1 + (0+0+0) × 2 + (0+0+0) = 6
- F(A4)=(2+2) ×1 + (0+0) × 2 + (0+0) × 2 = 4
- F(A5)=(2+2+2+2) ×1 + (0+0+0) × 2 + (0+0+0) = 8
- F(A6)=(2+2) ×1 + (0+0) × 2 + (0+0) = 4
- F(A7)=(2+2+2) ×1 + (0+0+0) × 2 + (0+0+0) = 6
- F(A8)=(2+2) ×1 + (0+0) × 2 + (0+0) × 2 = 4
- F(A9)=(2+2+2) ×1 + (0+0+0) × 2 + (0+0+0) × 2 = 6

最佳解優先搜尋策略(best-first search)

- 因此電腦選擇A5；若對手選擇A1，則盤面狀況如下：

X	A2	A3
A4	O	A6
A7	A8	A9

最佳解優先搜尋策略(best-first search)

依同樣的評估函數，可算出

- F(A2)=(1+1) ×1 + (1+0) × 2 + (-1+0) = 3
- F(A3)=(1+1+2) ×1 + (0+1+0) × 2 + (-1+0+0) = 5
- F(A4)=(1+1) ×1 + (1+0) × 2 + (0-1) = 3
- F(A6)=(1+2) ×1 + (1+0) × 2 + (0+0) = 5
- F(A7)=(1+1+2) ×1 + (0+1+0) × 2 + (-1+0+0) = 5
- F(A8)=(1+2) ×1 + (1+0) × 2 + (0+0) = 5
- F(A9)=(0+2+2) ×1 + (0+1+0) × 2 + (0-1+0) = 5

最佳解優先搜尋策略(best-first search)

- 由此可知，A3、A6、A7、A8 及A9都是不錯的選擇，假設電腦選擇A3；若對手選擇A7，則盤面狀況如下：

X	A2	O
A4	O	A6
X	A8	A9

最佳解優先搜尋策略(best-first search)

依同樣的評估函數，可算出

- F(A2)=(0+1) ×1 + (1+1) × 2 + (-1+0) = 4
- F(A4)=(0+1) ×1 + (0+1) × 2 + (4+0) = 7
- F(A6)=(1+1) ×1 + (1+1) × 2 + (0+0) = 6
- F(A8)=(1+1) ×1 + (0+1) × 2 + (-1+0) = 3
- F(A9)=(1+0+1) ×1 + (0+1+1) × 2 + (-1+0+0) = 5

最佳解優先搜尋策略(best-first search)

因此電腦選擇A4；若對手選擇A6，則盤面狀況如下：

X	A2	O
O	O	X
X	A8	A9

最佳解優先搜尋策略(best-first search)

依同樣的評估函數，可算出

- F(A2)=(0+1) ×1 + (1+1) × 2 + (-1+0) = 4
- F(A8)=(1+1) ×1 + (0+1) × 2 + (-1+0) = 3
- F(A9)=(1+0+0) ×1 + (0+1+1) × 2 + (-1+0-1) = 3

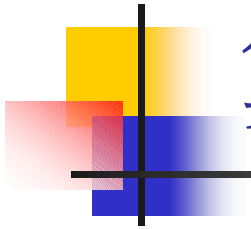
最佳解優先搜尋策略(best-first search)

因此電腦選擇A2；若對手選擇A8，則盤面狀況如下：

X	O	O
O	O	X
X	X	A9

- 最後電腦選擇A9，雙方以合局結束。

概念啓發式(Meta Heuristic)演算法



- 引導搜尋程序的策略
  - 簡單的區域搜尋方法
  - 複雜的學習程序
  - 以廣域搜尋的技巧避開陷入區域最佳解
- 例如：基因演算法（Genetic Algorithms）與禁忌搜尋法（Tabu Search）。

概念啓發式搜尋策略示意圖