人工智慧概論
Introduction to AI

第3章 問題求解與搜尋

蘇維宗 (Wei-Tsung Su)
suwt@au.edu.tw
564D

歷史版本

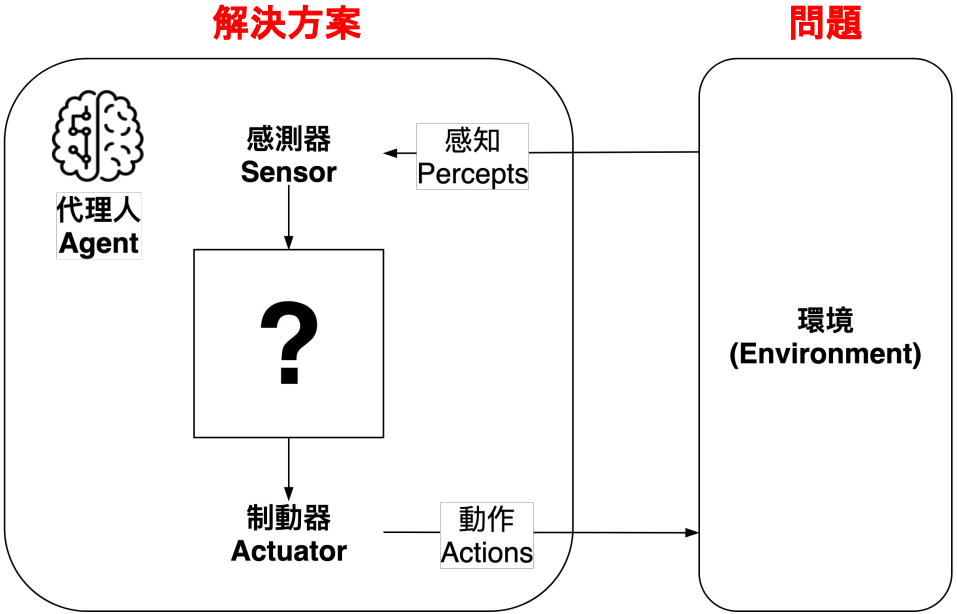
版本	說明	日期	負責人
v1.0	初版	2020/02/14	蘇維宗
v1.1	加入貪婪最佳優先搜尋範例	2020/04/10	蘇維宗
v1.2	加入爬山法	2020/05/10	蘇維宗

問題解決(Problem Solving)

前情提要

設計與實現代理人程式是為了解決環境中的**問題**。

那麼，要麼可以怎麼定義在現實環境中要讓代理人程式來解決的問題?



定義代理人程式所要解決的問題

可以用以下幾個項目來定義問題

- **初始狀態**　描述代理人的起始狀態
- **動作集合**　描述代理人可以選擇的動作集合
- **轉換模型**　描述代理人執行的動作(執行前狀態、行動、執行後狀態)
- **目標測試**　確認給定的狀態是否為目標狀態的方法
- **路徑成本**　達到目標狀態前所執行的所有動作所帶來的成本

Path to Bucharest問題

- **初始狀態**
  - *In(Arad)*
- 動作集合
- 轉換模型
- 目標測試
- 路徑成本

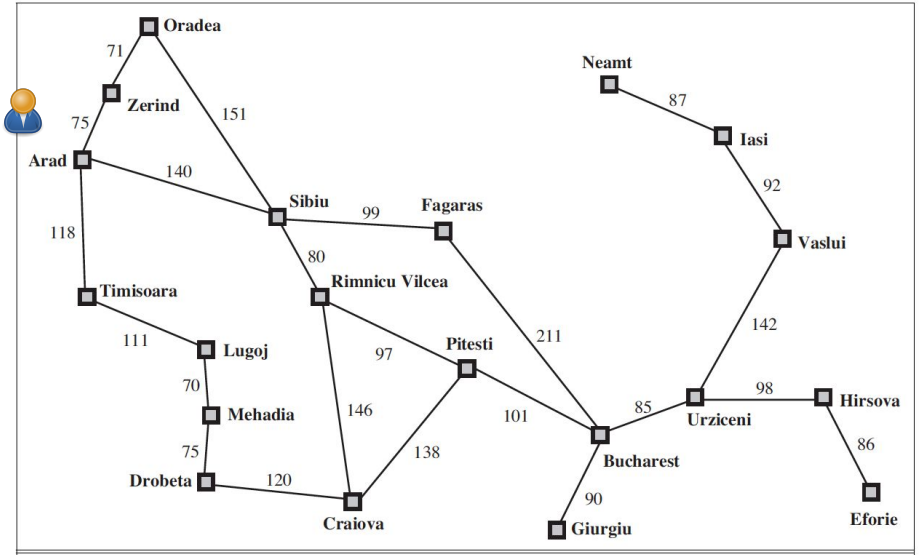


Figure 3.2 A simplified road map of part of Romania.

Path to Bucharest問題(續)

- 初始狀態
- **動作集合**
  - *Go(Zerind)*
  - *Go(Sibiu)*
  - *Go(Timisoara)*
- 轉換模型
- 目標測試
- 路徑成本

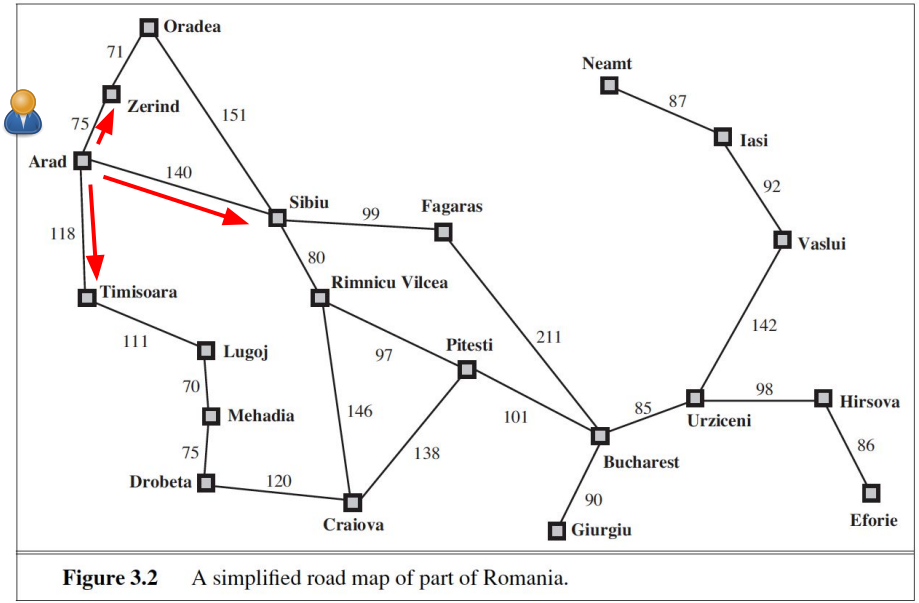


Figure 3.2 A simplified road map of part of Romania.

Path to Bucharest問題(續)

- 初始狀態
- 動作集合
- **轉換模型**
  - **RESULT**(
    In(Arad),
    Go(Sibiu)
) = *In(Sibiu)*
- 目標測試
- 路徑成本

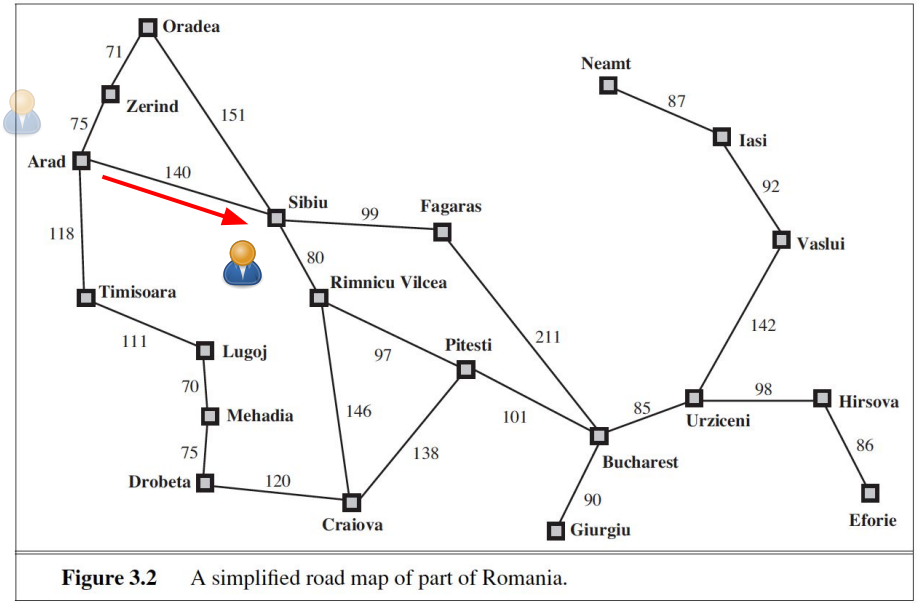


Figure 3.2 A simplified road map of part of Romania.

Path to Bucharest問題(續)

- 初始狀態
- 動作集合
- 轉換模型
- **目標測試**
  - *In(Bucharest)*
- 路徑成本

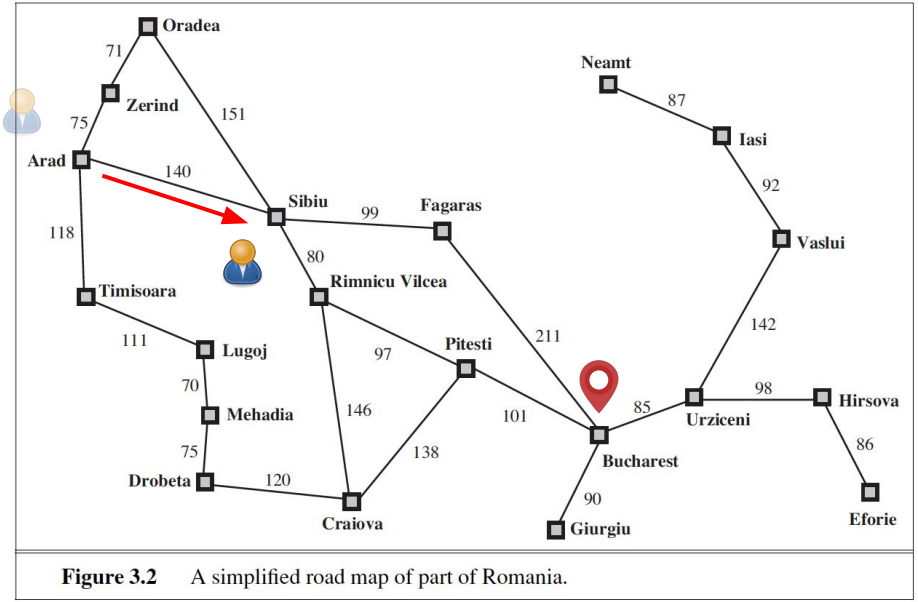


Figure 3.2 A simplified road map of part of Romania.

Path to Bucharest問題(續)

- 初始狀態
- 動作集合
- 轉換模型
- 目標測試
- **路徑成本**
  - C(Arad>Sibiu) = 140
  - C(Sibiu>Rimnicu Vilcea) = 80
  - C(Rimnicu Vilcea>Pitesti) = 97
  - C(Pitesti>Bucharest) = 101
  - Path Cost = 140+80+97+101 = 418

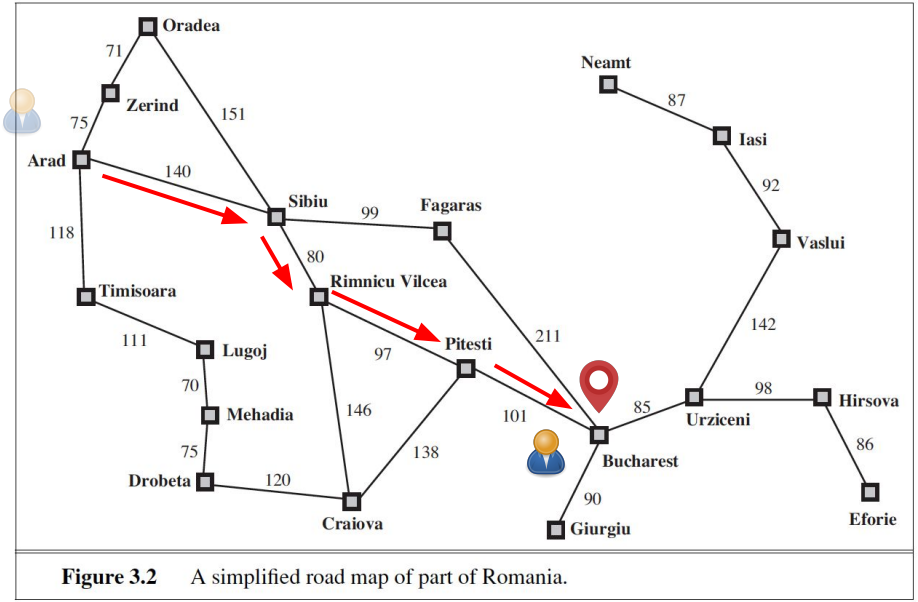


Figure 3.2 A simplified road map of part of Romania.

練習：掃地機器人(1x2)

- **狀態集合**：所在位置(2) * 位置A是否乾淨(2) * 位置B是否乾淨(2) = 8種
- **初始狀態**：8種的其中一種
- **動作集合**：Left、Right、Clean
- **轉換模型**：如後圖
- **目標測試**：檢查所有位置都是乾淨
- **路徑成本**：走一步成本為1，路徑成本為走的步數

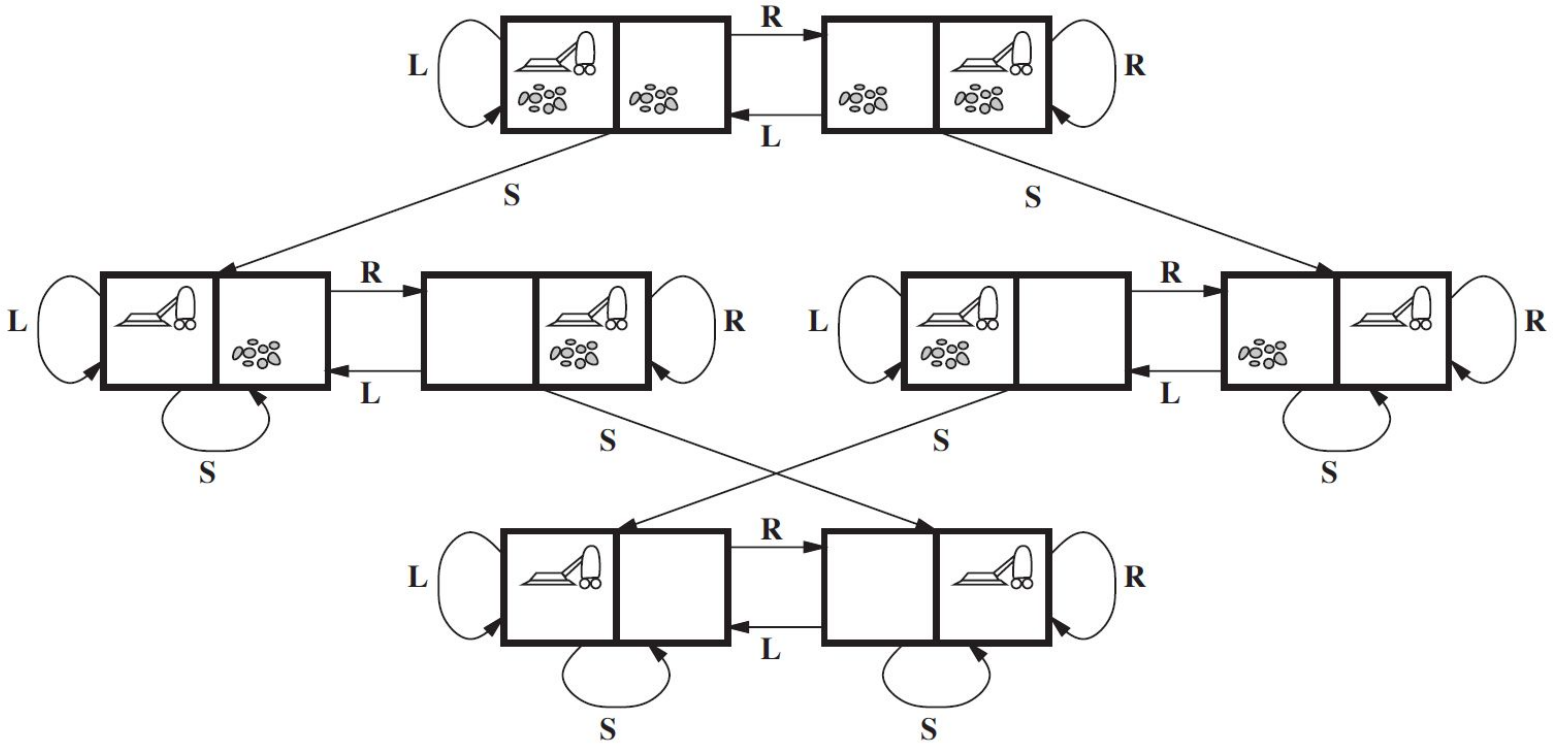
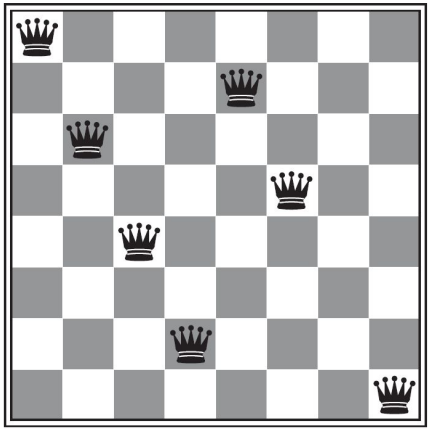


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

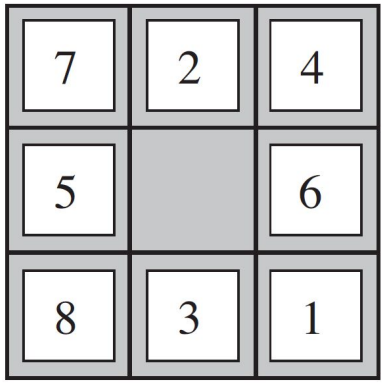
8皇后問題

- **可能狀態**:8個皇后在任何一個可能的位置上(有幾種?)
- **初始狀態**:沒有皇后在板子上
- **動作集合**:新增一個皇后在板子上
- **轉換模型**:新增一個皇后之後的板子狀態
- **目標測試**:8個在板子上的皇后且不會互相攻擊
- **路徑成本**:無

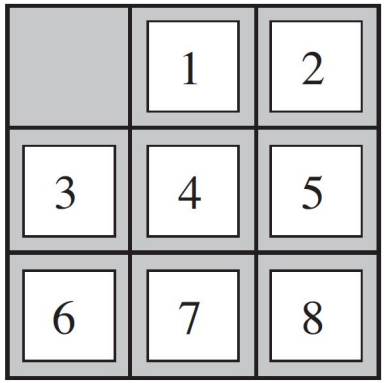


練習3-1

試著描述8-puzzle問題



Start State



Goal State

搜尋解答(Searching)

搜尋解決方案

定義好問題後，接著要擬定一個**搜尋策略(search strategy)**從初始狀態開始根據轉換模型形成一個**搜尋樹(search tree)**來搜尋解決方案。

以Path to Bucharest問題為例來說明如何以搜尋策略建構搜尋樹。

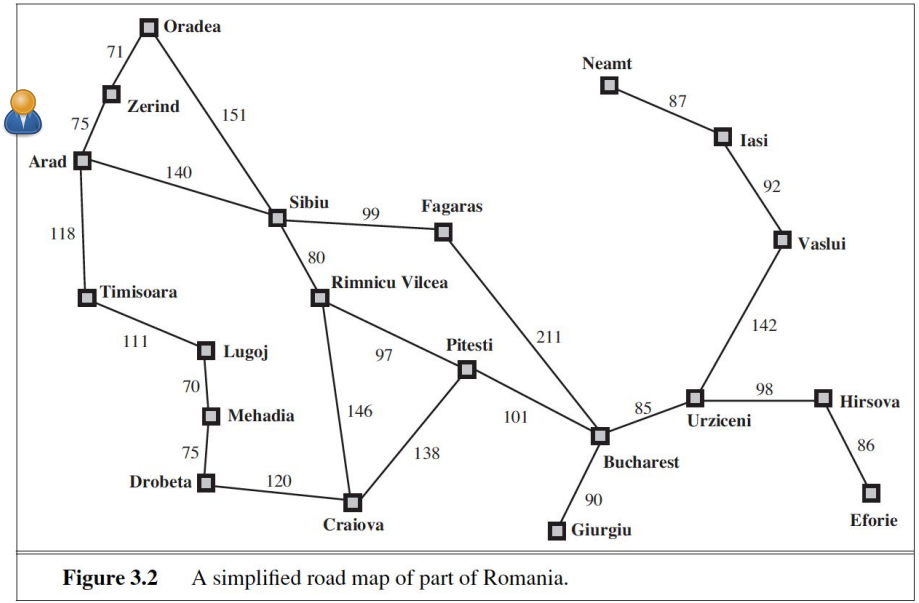
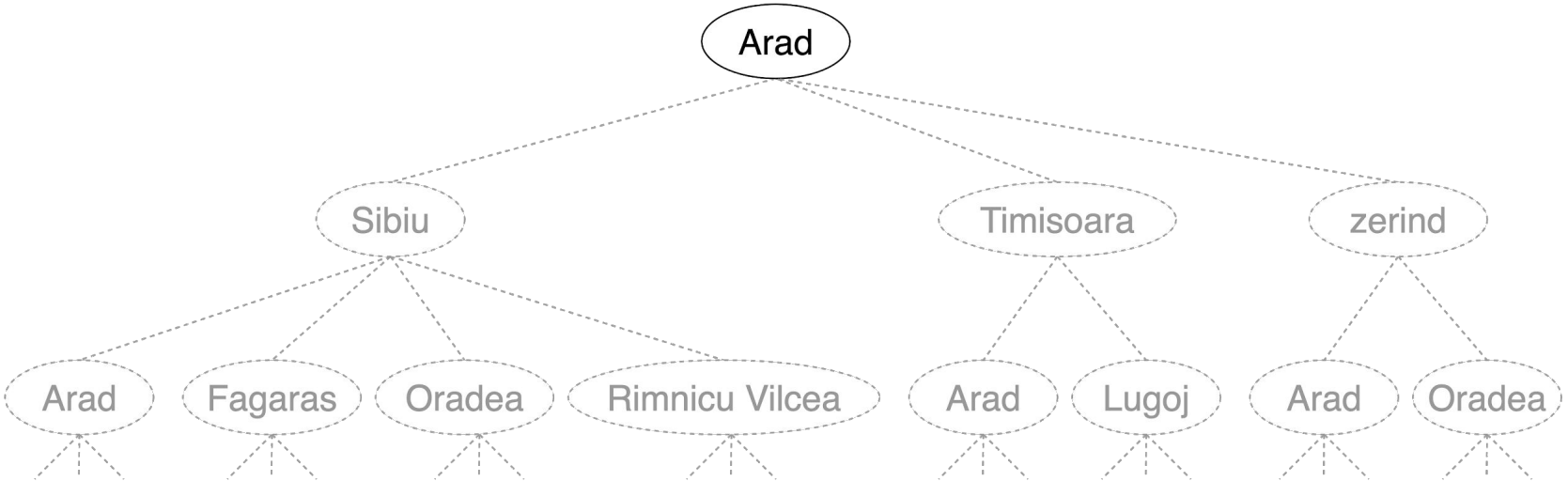
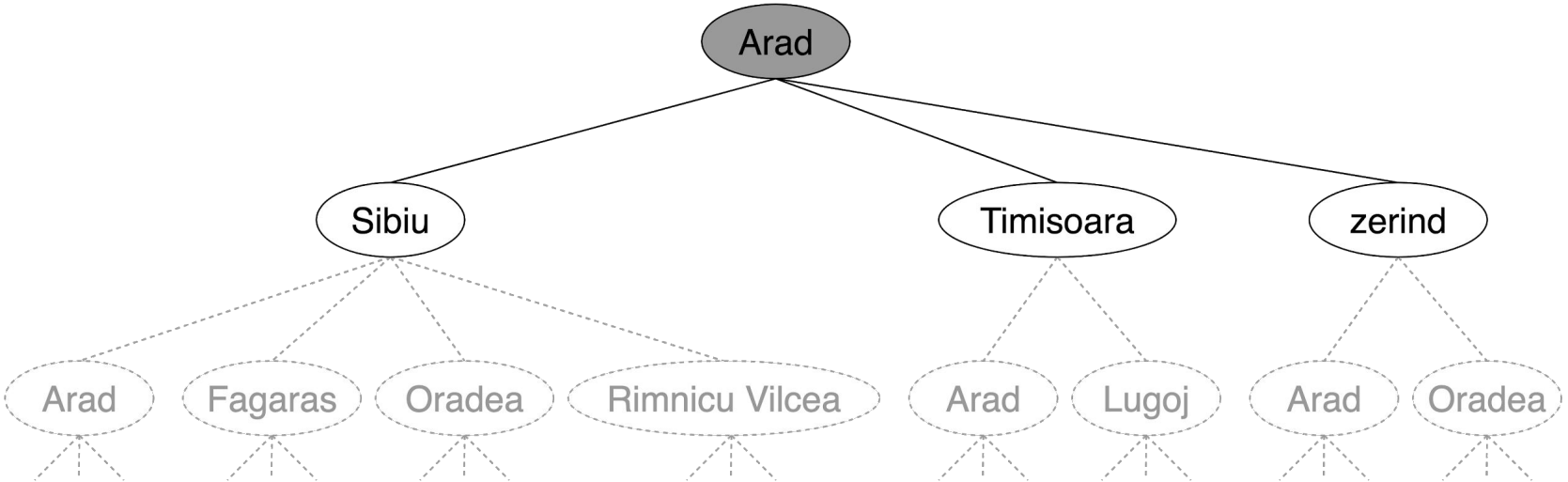


Figure 3.2 A simplified road map of part of Romania.

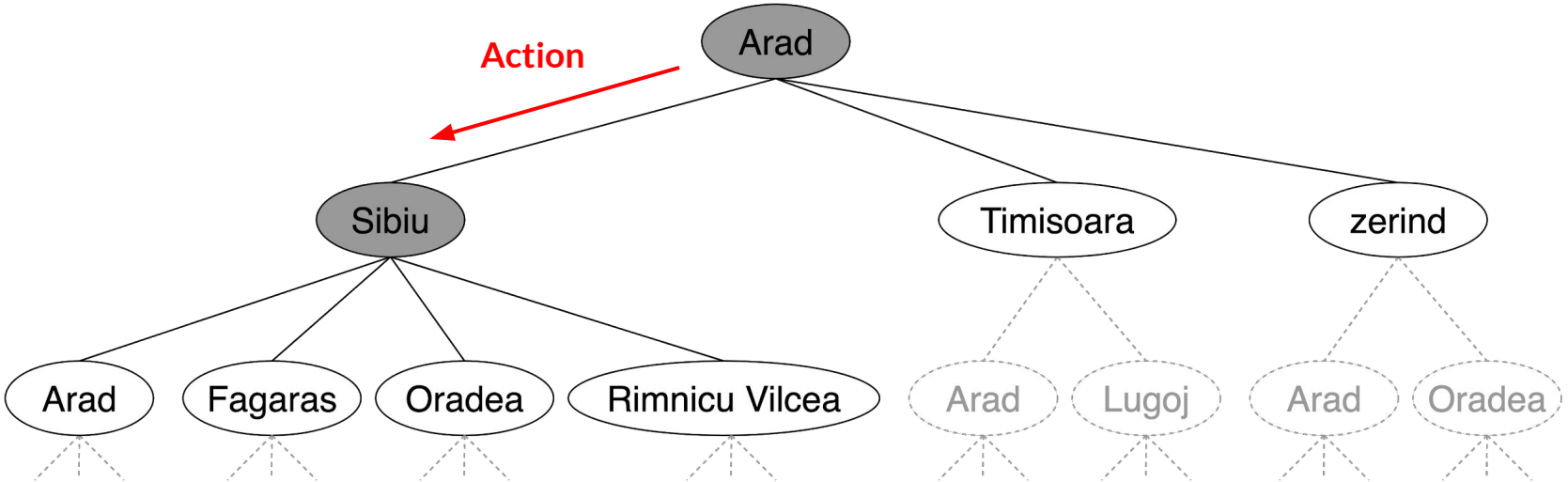
Path to Bucharest問題(初始狀態 / Arad)



Path to Bucharest問題(搜尋策略 / 3種選擇)



Path to Bucharest問題(搜尋策略 / 6種選擇)



解決方案的效能

除了找到搜尋策略之外，搜尋策略的效能評估也很重要。

- **完整性**: 是否可以找到解決方案?
- **最佳解**: 找到的是否為最佳解?
- **時間複雜度**: 要花多少時間找到解決方案?
- **空間複雜度**: 要花多少記憶體找到解決方案?

無資訊的搜尋策略

無資訊的搜尋策略

無資訊的搜尋策略是只根據問題的定義來搜尋解決方案，又稱為盲目搜尋(blind search)。例如

- 廣度優先搜尋(Breadth First Search，簡稱 BFS)
- 深度優先搜尋(Depth First Search，簡稱DFS)
- 統一成本搜尋(Uniform Cost Search)
- ...

廣度優先搜尋(Breadth First Search / BFS)

BFS是利用佇列(FIFO)來決定搜尋的順序

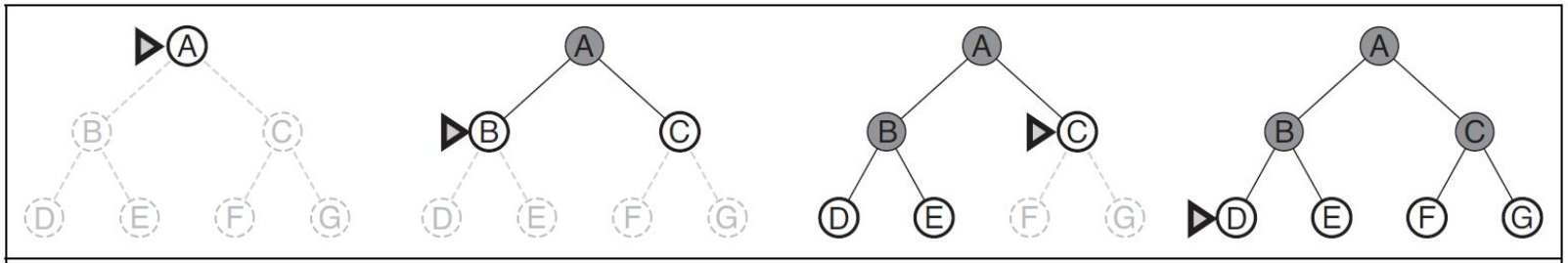
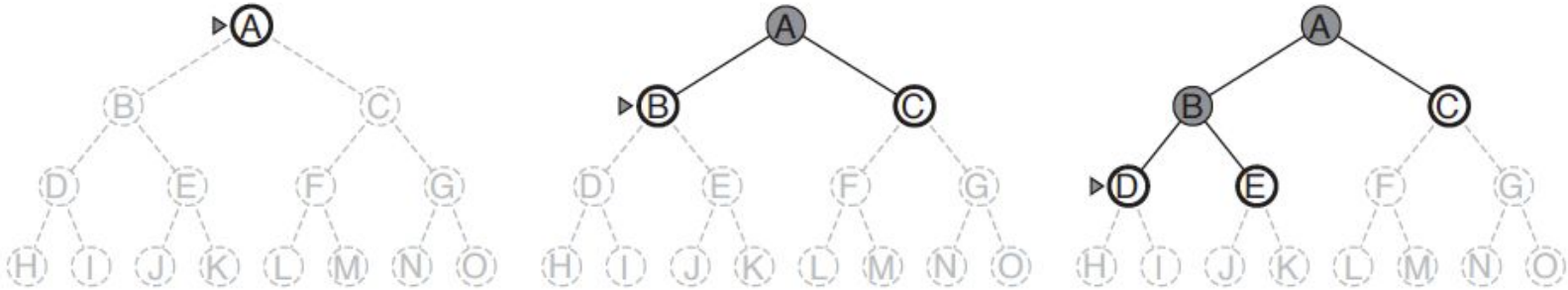


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

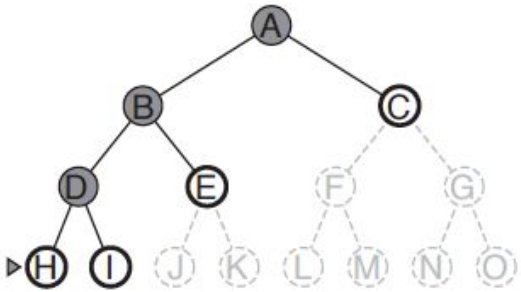
深度優先搜尋(Depth First Search / DFS)

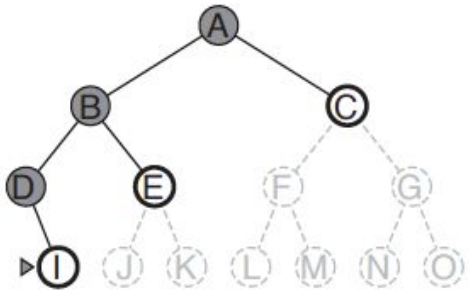
DFS是利用堆疊(LIFO)來決定搜尋順序

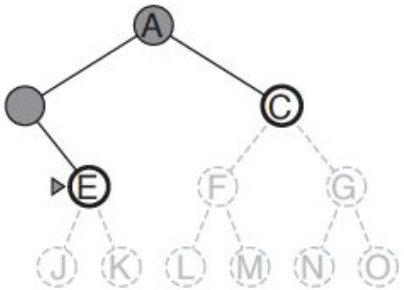


深度優先搜尋(Depth First Search / DFS)

DFS是利用堆疊(LIFO)來決定搜尋順序

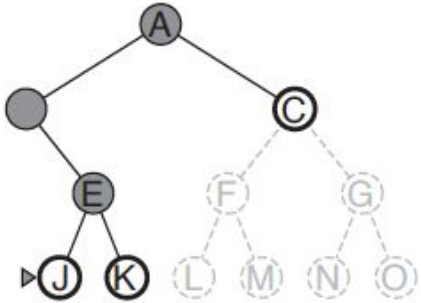


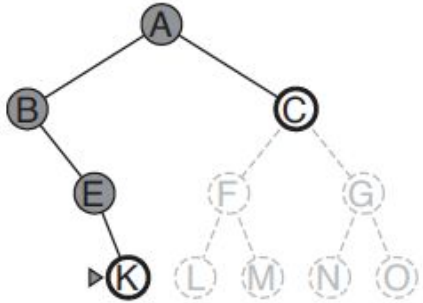


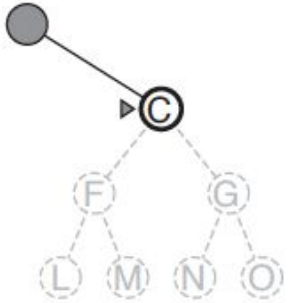


深度優先搜尋(Depth First Search / DFS)

DFS是利用堆疊(LIFO)來決定搜尋順序

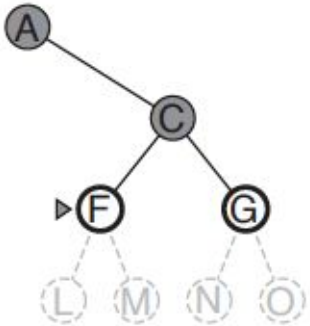


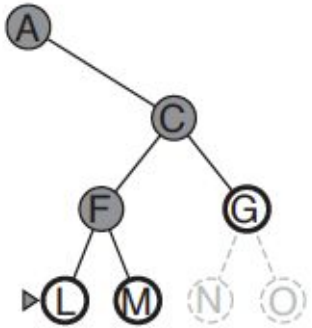


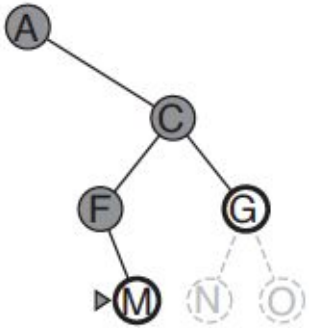


深度優先搜尋(Depth First Search / DFS)

DFS是利用堆疊(LIFO)來決定搜尋順序

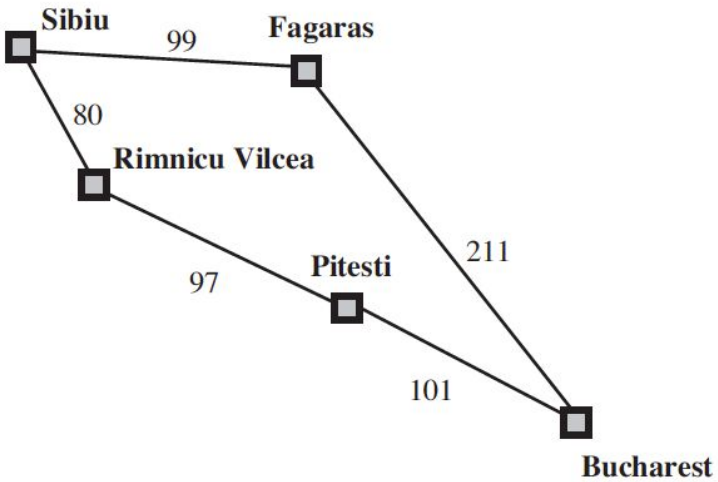






統一成本搜尋(Uniform Cost Search / UCS)

UCS是利用優先佇列(priority queue)來決定搜尋的順序



有資訊(啟發式的)的搜尋策略

有資訊的搜尋策略

利用對問題的特定知識(domain knowledge)來搜尋解答。例如

- 貪婪最佳優先搜尋(Greedy Best First Search)
- A*搜尋
- ...

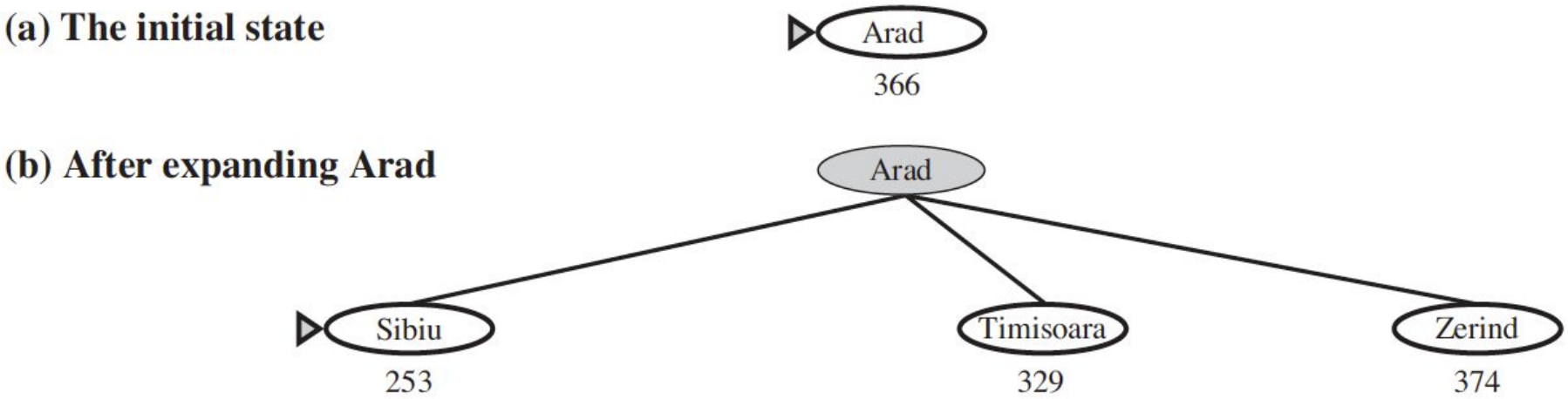
貪婪最佳優先搜尋(Greedy Best First Search)

假設已知每個城市到Bucharest的直線最短距離

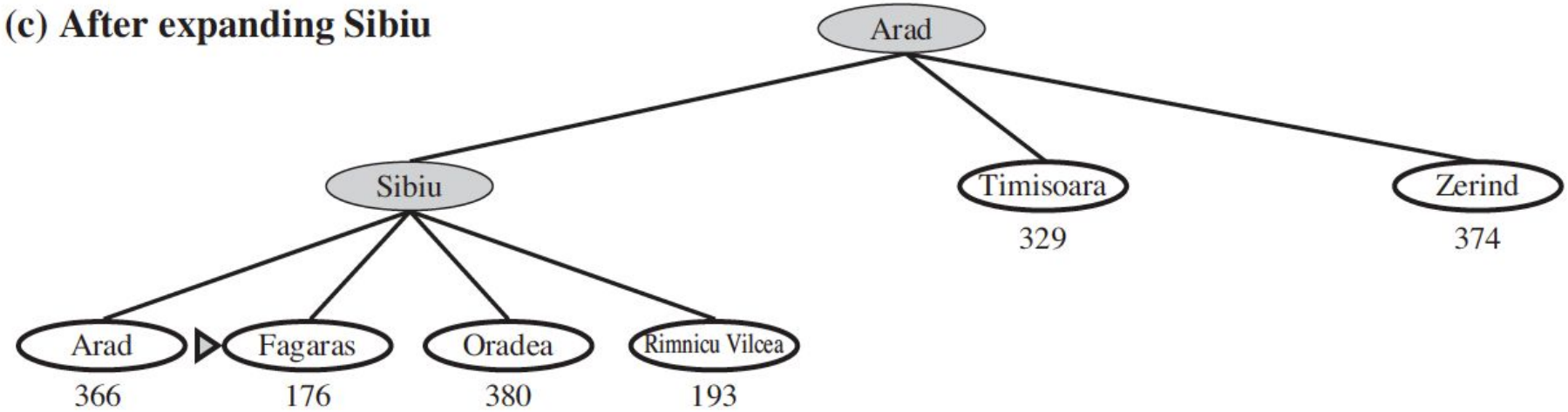
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

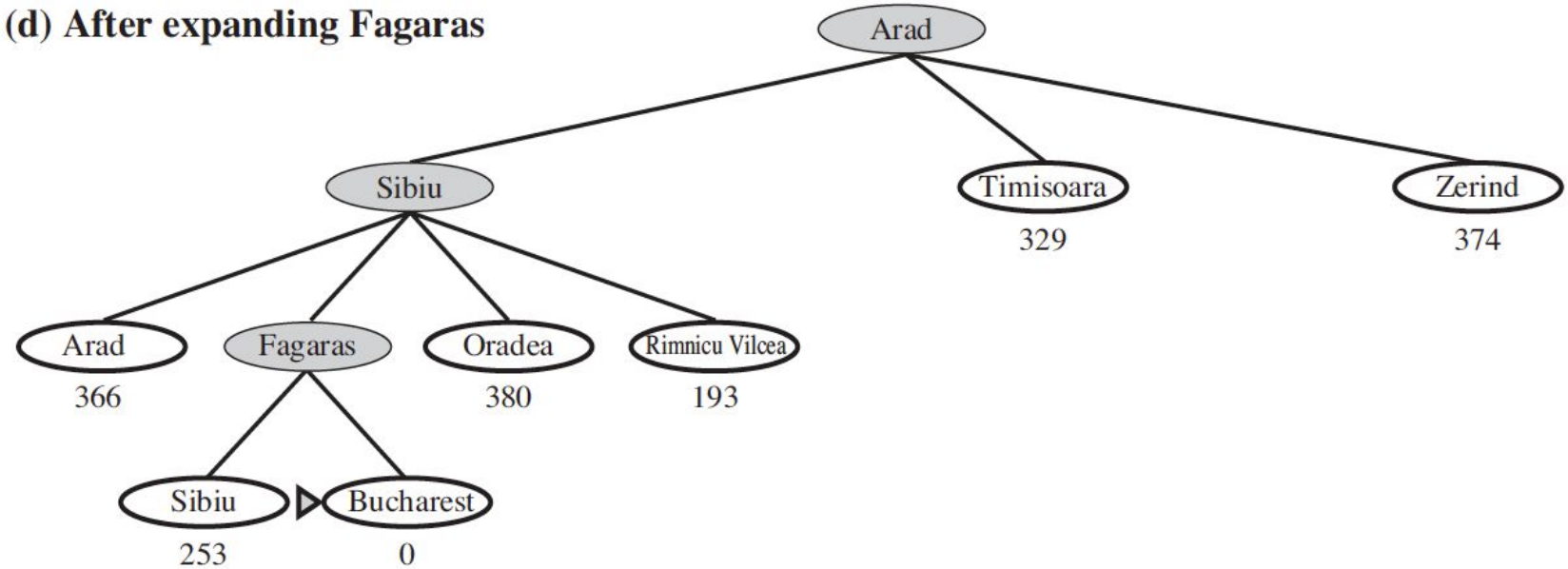
貪婪最佳優先搜尋(Greedy Best First Search)



貪婪最佳優先搜尋(Greedy Best First Search)



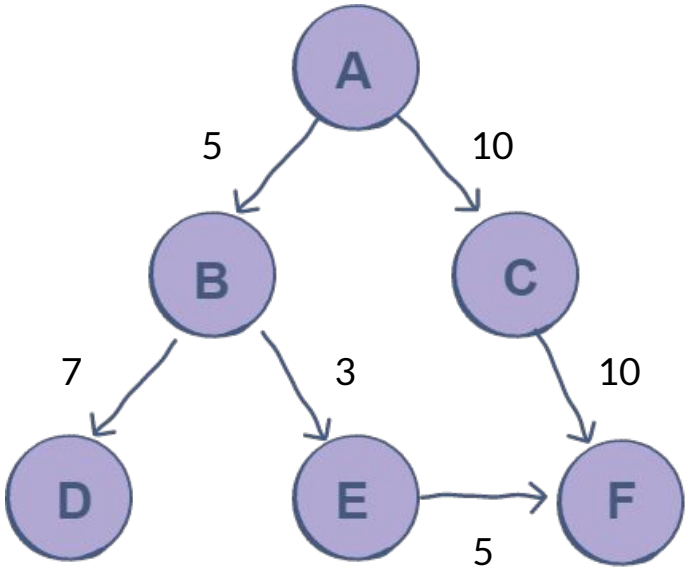
貪婪最佳優先搜尋(Greedy Best First Search)



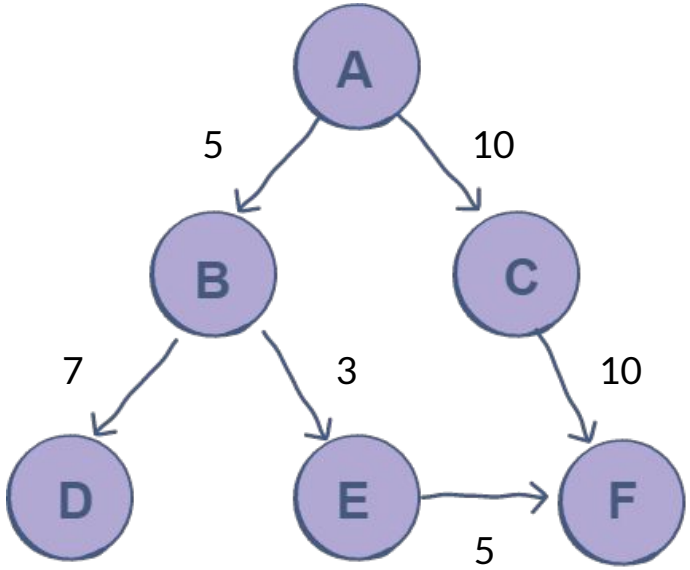
複習: 搜尋解答

請寫出此圖的各種搜尋結果

1. 廣度優先搜尋(BFS)
   a. 從A開始遍歷(traverse)所有節點的順序
   b. 從A到F的路徑
2. 深度優先搜尋(DFS)
   a. 從A開始遍歷(traverse)所有節點的順序
   b. 從A到F的路徑
3. 統一成本搜尋(UCS)
   a. 從A開始遍歷(traverse)所有節點的順序
   b. 從A到F的路徑



請寫出此圖的各種搜尋結果(續)



撰寫Python程式執行BFS與DFS

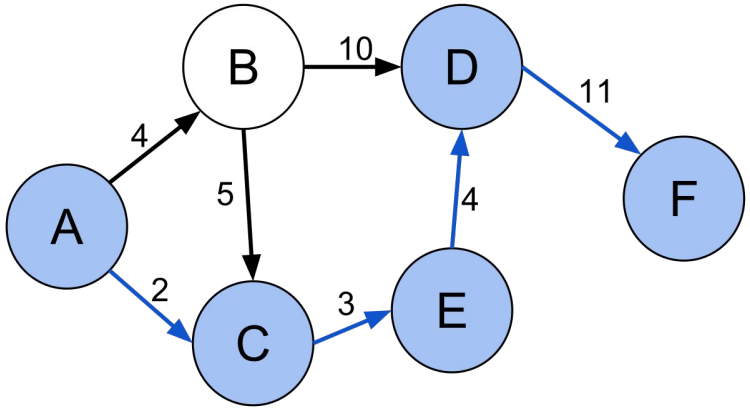
資料結構

```
graph = {
  'A' : [{'node':'B','cost':1},{'node':'C','cost':1}],
  'B' : [{'node':'D','cost':1}, {'node':'E','cost':1}],
  'C' : [{'node':'F','cost':1}],
  'D' : [],
  'E' : [{'node':'F','cost':1}],
  'F' : []
}
```

考試: 搜尋解答

請寫出此圖的各種搜尋結果

1. 廣度優先搜尋(BFS)
   a. 從A開始遍歷(traverse)所有節點的順序
   b. 從A到F的路徑
2. 深度優先搜尋(DFS)
   a. 從A開始遍歷(traverse)所有節點的順序
   b. 從A到F的路徑
3. 統一成本搜尋(UCS)
   a. 從A開始遍歷(traverse)所有節點的順序
   b. 從A到F的路徑

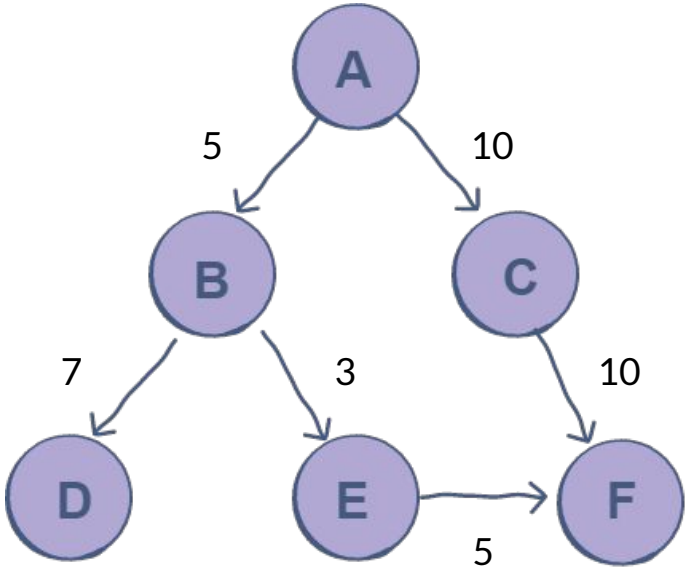


作業3-1

撰寫Python程式執行UCS

撰寫Python程式執行UCS

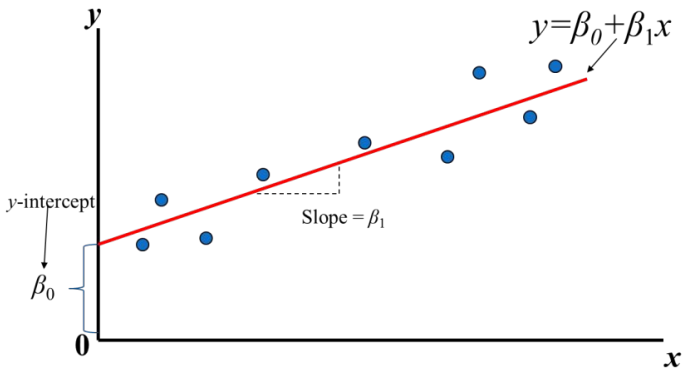
繳交期限: 05/11/2020
繳交方式: ipynb檔案



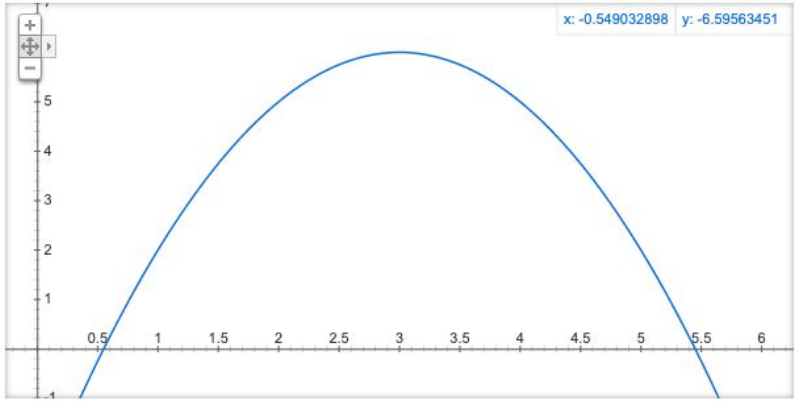
目標狀態未知的搜尋

目標狀態未知的搜尋問題

有時候，待解決問題的目標狀態是未知的。例如，



尋找一個直線與平面上的資料距離最近



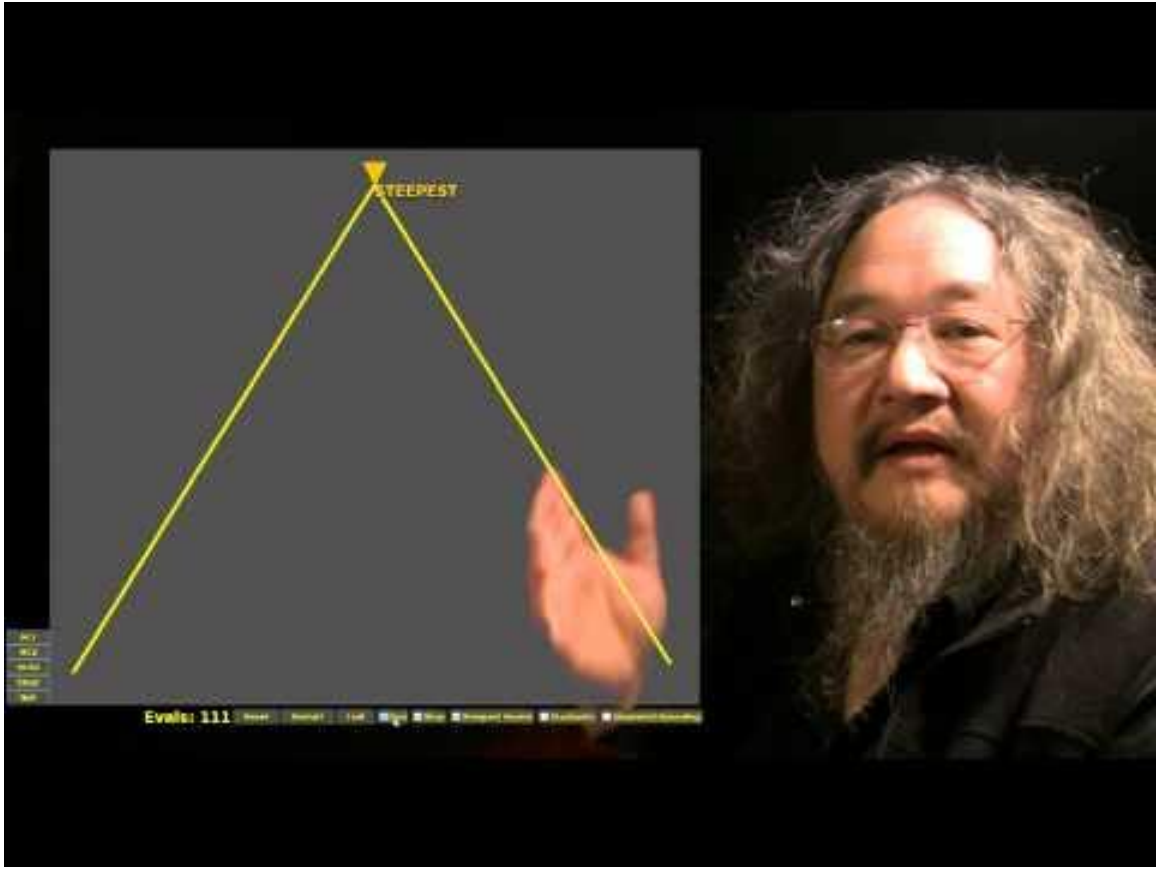
尋找函數的極值

迭代搜尋演算法(Iterative Searching)

迭代式搜尋演算法的步驟如下

1. 決定起始狀態 (initial value)
2. 重複改變狀態 (step function)
3. 接受或拒絕狀態(evaluation function)
4. 重複步驟2與3直到滿足停止條件(stop condition)

例如，爬山法(hill climbing)、模擬退火法(simulated annealing)都屬於此類型演算法。



爬山法(Hill Climbing)

爬山法範例

找到函數的最大值: $f(x) = -x^2 + 6x - 3$

1. **決定起始狀態 (initial value)**
   a. 可以是0、隨機數 (或其它方式)
2. 重複改變狀態 (step function)
3. 接受或拒絕狀態(evaluation function)
4. 重複步驟2與3直到滿足停止條件(stop condition)

爬山法範例

找到函數的最大值: $f(x) = -x^2 + 6x - 3$

1. 決定起始狀態 (initial value)
2. **重複改變狀態 (step function)**
   a. 讓x每次增加x'=x+0.1與減少x'=x-0.1 (或其它方式)
3. 接受或拒絕狀態(evaluation function)
4. 重複步驟2與3直到滿足停止條件(stop condition)

爬山法範例

找到函數的最大值: $f(x) = -x^2 + 6x - 3$

1. 決定起始狀態 (initial value)
2. 重複改變狀態 (step function)
3. **接受或拒絕狀態(evaluation function)**
   a. 如果$f(x')$比$f(x)$的的結果好就接受新狀態
4. 重複步驟2與3直到滿足停止條件(stop condition)

爬山法範例

找到函數的最大值: $f(x) = -x^2 + 6x - 3$

1. 決定起始狀態 (initial value)
2. 重複改變狀態 (step function)
3. 接受或拒絕狀態(evaluation function)
4. 重複步驟2與3直到滿足停止條件(stop condition)
    a. 當沒有$f(x')$比$f(x)$好時或當測試了1,000次後結束並回傳結果

作業3-2

撰寫爬山法程式(Python)找出

$f(x) = -x^3 + 6x + 10$

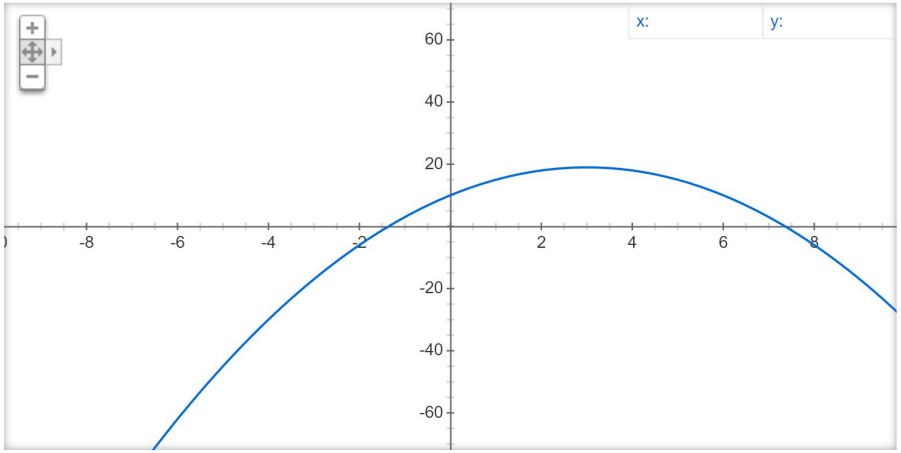
其中 $x >= 0$

的最大值

1. 想辦法讓答案盡量接近最佳解
2. 想辦法讓測試的次數變少
3. 想辦法讓函式可代容易替換

繳交期限: 05/25/2020

繳交方式: ipynb檔案



Q & A

Computer History Museum, Mt. View, CA