第四週建議作業

- 我們在上課的時候一直強調，A* 要發揮最佳的功效，必須先確定 heuristic function 是 admissible。我們沒有很多時間詳細說明，萬一 heuristic function 其實不符合 admissibility 的條件的話，會有甚麼後果。以下是兩個例題。

- 假定我們採用圖 3.22 的 heuristic function，但是假設我們實際上高估了 Pitesti 到 Bucharest 的距離，假定我們估計的距離不是圖 3.22 的100，而是131。在這一些條件之下，假定我們要應用 A* 演算法找尋圖 3.2 上從Arad 到 Bucharest 的最短路線。繪製類似課本圖 3.24 的 search tree，並且參考圖 3.7 的說法，我們一共會執行幾次的expand the chosen node 的動作？依序是試探了哪一些路線(參考圖3.8)？你找到的答案是否是最短路線？
  - 以上答案跟我們另 h(Pitesti)=100 的時候，有何不同？

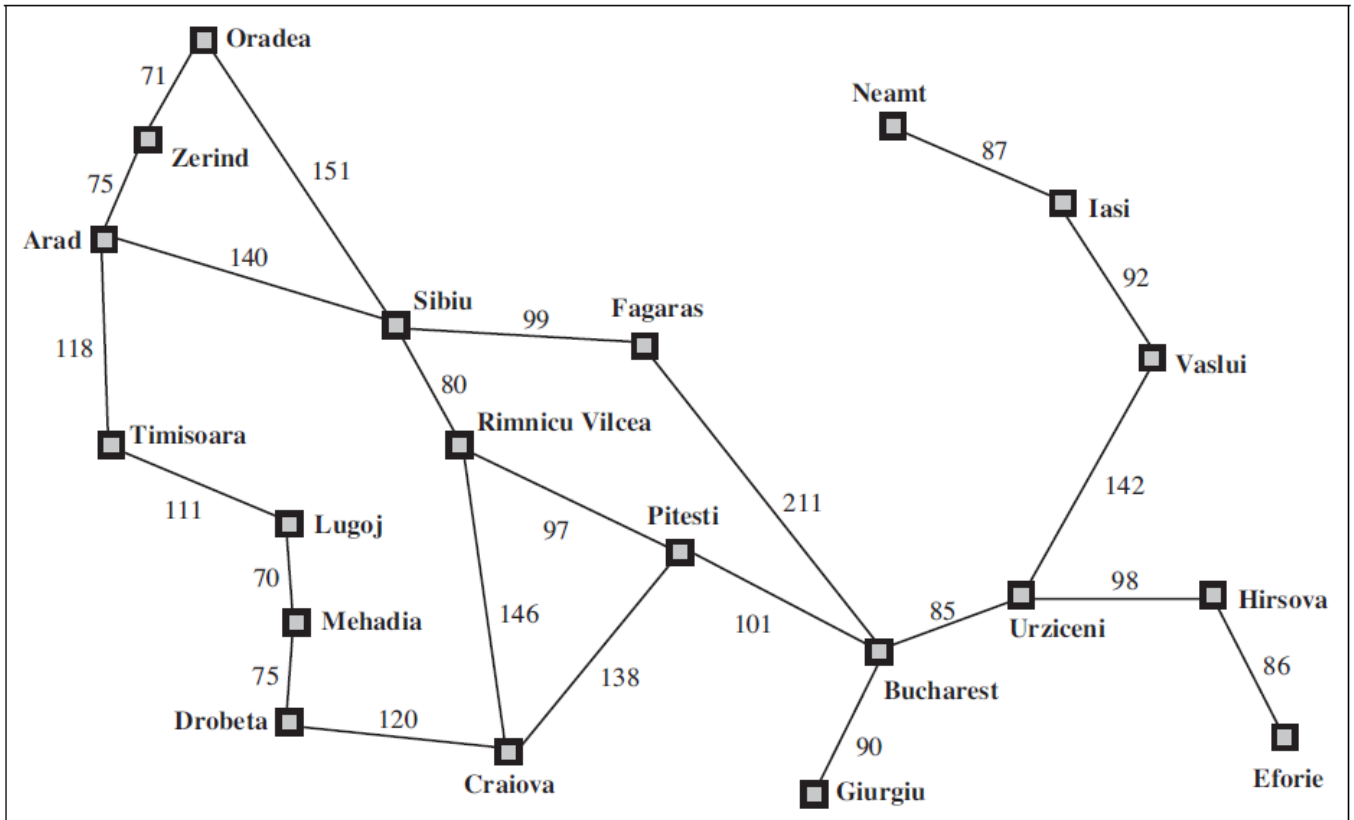
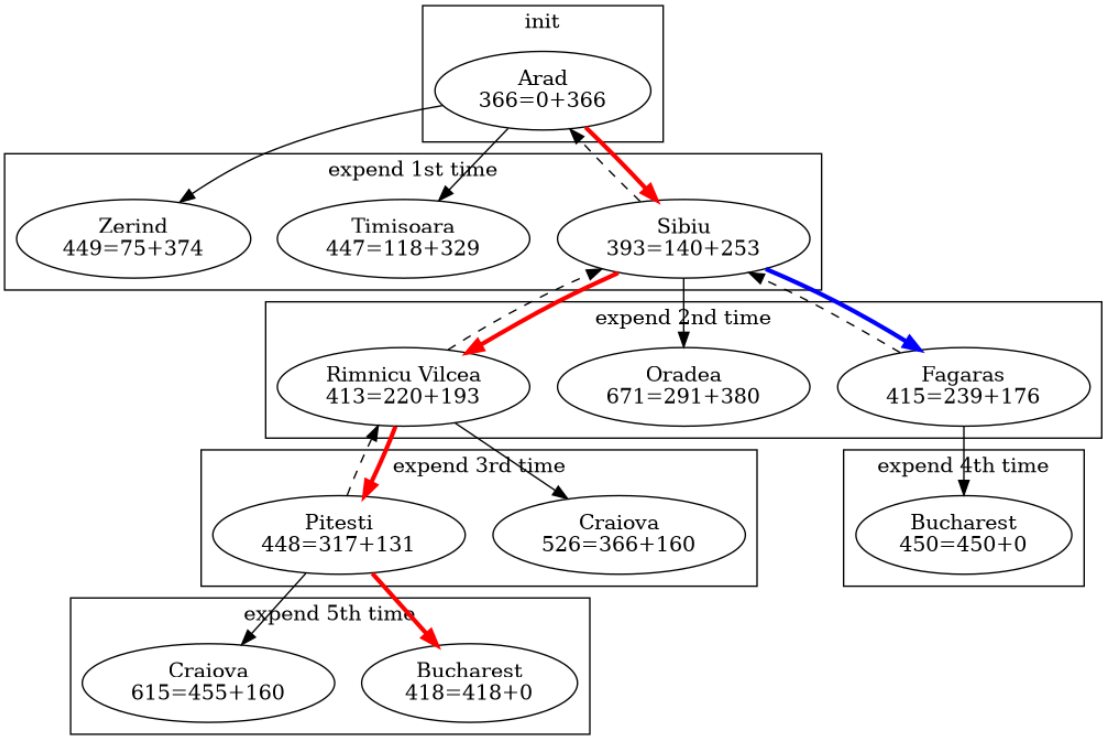


Figure 3.2 A simplified road map of part of Romania.

In [28]:

```
digraph {
    subgraph cluster_0 {
        label = "init";
        A [label = "Arad\n366=0+366"];
    }

    subgraph cluster_1 {
        label = "expend 1st time";

        S [label = "Sibiu\n393=140+253"];
        T [label = "Timisoara\n447=118+329"];
        Z [label = "Zerind\n449=75+374"];

        A -> S [color = red, penwidth = 3.0];
        A -> T;
        A -> Z;
    }

    subgraph cluster_2 {
        label = "expend 2nd time";

        R [label = "Rimnicu Vilcea\n413=220+193"];
        O [label = "Oradea\n671=291+380"];
        F [label = "Fagaras\n415=239+176"];

        S -> A [style = "dashed"];
        S -> F [color = blue, penwidth = 3.0];
        S -> O;
        S -> R [color = red, penwidth = 3.0];
    }

    subgraph cluster_3 {
        label = "expend 3rd time";

        C [label = "Craiova\n526=366+160"];
        P [label = "Pitesti\n448=317+131"];

        R -> C;
        R -> P [color = red, penwidth = 3.0];
        R -> S [style = "dashed"];
    }

    subgraph cluster_4 {
        label = "expend 4th time";

        B [label = "Bucharest\n450=450+0"]

        F -> S [style = "dashed"];
        F -> B;
    }

    subgraph cluster_5 {
        label = "expend 5th time";

        B2 [label = "Bucharest\n418=418+0"];
        C2 [label = "Craiova\n615=455+160"]

        P -> B2 [color = red, penwidth = 3.0];
        P -> C2;
```

```
        P -> R [style = "dashed"];
    }
}
```



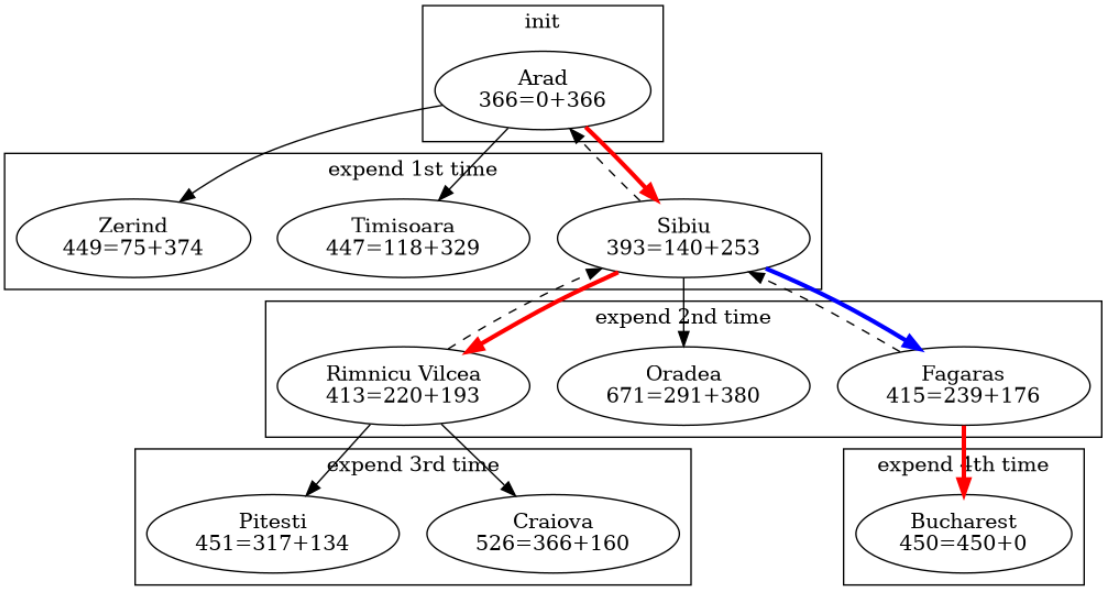
回答結果

1. 會執行 5 次 expand the chosen node
2. 找了 A -> S -> R -> P -> B
3. 找到的答案是最短路徑
4. 此結果與 h = 100 時相同

- 假定我們採用圖 3.22 的 heuristic function，但是假設我們實際上高估了 Pitesti 到 Bucharest 的距離，假定我們估計的距離不是圖 3.22 的100，而是134。在這一些條件之下，假定我們要應用 A* 演算法找尋圖 3.2 上從Arad 到 Bucharest 的最短路線。繪製類似課本圖 3.24 的 search tree，並且參考圖 3.7 的說法，我們一共會執行幾次的expand the chosen node 的動作？依序是試探了哪一些路線(參考圖3.8)？你找到的答案是否是最短路線？
    - 以上答案跟我們另 h(Pitesti)=100 的時候，有何不同？

In [27]:

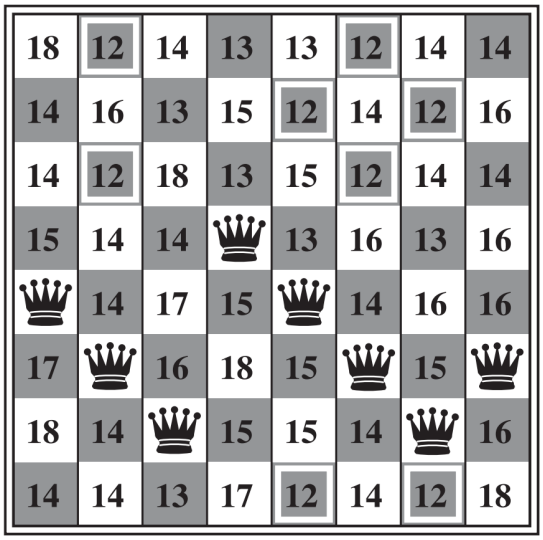
```
digraph {
    subgraph cluster_0 {
        label = "init";
        A [label = "Arad\n366=0+366"];
    }

    subgraph cluster_1 {
        label = "expend 1st time";

        S [label = "Sibiu\n393=140+253"];
        T [label = "Timisoara\n447=118+329"];
        Z [label = "Zerind\n449=75+374"];

        A -> S [color = red, penwidth = 3.0];
        A -> T;
        A -> Z;
    }

    subgraph cluster_2 {
        label = "expend 2nd time";

        R [label = "Rimnicu Vilcea\n413=220+193"];
        O [label = "Oradea\n671=291+380"];
        F [label = "Fagaras\n415=239+176"];

        S -> A [style = "dashed"];
        S -> F [color = blue, penwidth = 3.0];
        S -> O;
        S -> R [color = red, penwidth = 3.0];
    }

    subgraph cluster_3 {
        label = "expend 3rd time";

        C [label = "Craiova\n526=366+160"];
        P [label = "Pitesti\n451=317+134"];

        R -> C;
        R -> P;
        R -> S [style = "dashed"];
    }

    subgraph cluster_4 {
        label = "expend 4th time";

        B [label = "Bucharest\n450=450+0"]

        F -> S [style = "dashed"];
        F -> B [color = red, penwidth = 3.0];
    }
}
```

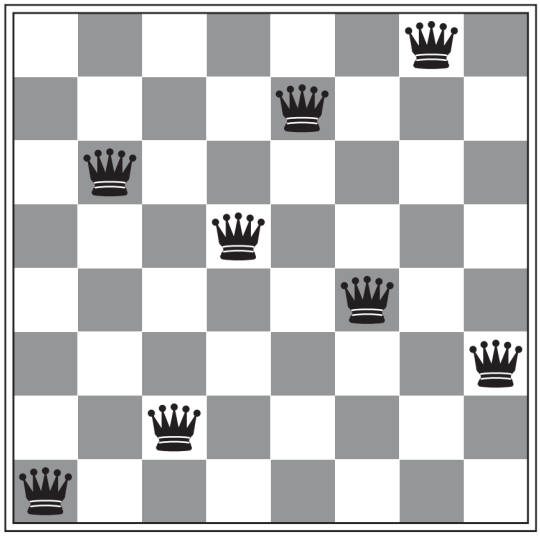


回答結果

1. 會執行 4 次 expand the chosen node
2. 找了 A -> S -> F -> B
3. 找到的答案不是最短路徑
4. 此結果與 h = 100 時不同

- 檢驗 AIMA 圖4.3 標題裡面說 h=17 是否正確？





(a) (b)

Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

In [4]:

```
;;  假設從左上開始,點位為 (0 0),往右為第一位,右上為 (7 0),往下為第二位,左下為 (0 7),皇后的編號由左至右為 0 ~ 7

(require racket/list
         racket/function)

(define q0 (cons 0 4))
(define q1 (cons 1 5))
(define q2 (cons 2 6))
(define q3 (cons 3 3))
(define q4 (cons 4 4))
(define q5 (cons 5 5))
(define q6 (cons 6 6))
(define q7 (cons 7 5))

(define queens (list q0 q1 q2 q3 q4 q5 q6 q7))

(define get-column
  (lambda (queen)
    (car queen)))

(define get-row
  (lambda (queen)
    (cdr queen)))

(define h-func
  (lambda (queens queen)
    (let loop ([col (get-column queen)]
               [row (get-row queen)]
               [qs queens]
               [h 0])
      (if (null? qs)
          h
          (if (= col 0)
              (loop col row (rest qs) h)

              (let* ([current (first qs)]
                     [curr-col (get-column current)]
                     [curr-row (get-row current)])
                (if (= col curr-col)
                    h  ;; 只算對前面棋子的攻擊性,所以到自己就回傳

                    (if (or (= row curr-row)
                            (= (abs (- row curr-row))
                               (abs (- col curr-col))))
                        (loop col row (rest qs) (+ h 1))  ;; 有攻擊到前方棋子的情況

                        (loop col row (rest qs) h)  ;; 沒有攻擊到前方棋子的情況
))))))))


(define h-sum (foldl + 0
                     (map (curry h-func queens)
                          queens)))

(displayln (string-append "此棋盤的 h 為"
                          (number->string h-sum)))
```

此棋盤的 h 為17

- 試試看別人處理 8 queens 的程式
  - 例如：https://developers.google.com/optimization/cp/queens (https://developers.google.com/optimization/cp/queens)
  - 網路上有很多這一類的程式，可以用 8 queens problem 找找看
  - 他們都用一樣的觀點 (formulations) 來解這一個問題嗎？
    - 有人使用我們所講的暴力方法嗎？
    - 如果下載回家使用的話，那樣的程式可以處理多大的棋盤？
    - 程式要執行多久？
    - 有人使用 genetic algorithms 嗎？

In [2]:

```
## 一個簡單的 backtrack 搜尋法 (暴力法)
## 參：https://rosettacode.org/wiki/N-queens_problem#Python:_Simple_Backtracking_Solution_.28functional_style.29

def solve_func(n, i, a, b, c):
    if i < n:
        for j in range(n):
            if j not in a and i+j not in b and i-j not in c:
                for solution in solve_func(n, i+1, a+[j], b+[i+j], c+[i-j]):
                    yield solution
    else:
        yield a


def solve_first(n):
    solutions = solve_func(n, 0, [], [], [])
    return next(solutions)


print(solve_first(8))
```

```
[0, 4, 7, 5, 2, 6, 1, 3]
```

In [ ]:

```
import time

limit = 60

for i in range(8, 31):
    print("處理 {} queen".format(i))
    start = time.time()
    print("結果：{}".format(solve_first(i)))
    end = time.time()

    if limit < (end - start):
        print("於 i = {} 時超過 {} 秒".format(i, limit))
        break
```

```
處理 8 queen
結果：[0, 4, 7, 5, 2, 6, 1, 3]
處理 9 queen
結果：[0, 2, 5, 7, 1, 3, 8, 6, 4]
處理 10 queen
結果：[0, 2, 5, 7, 9, 4, 8, 1, 3, 6]
處理 11 queen
結果：[0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
處理 12 queen
結果：[0, 2, 4, 7, 9, 11, 5, 10, 1, 6, 8, 3]
處理 13 queen
結果：[0, 2, 4, 1, 8, 11, 9, 12, 3, 5, 7, 10, 6]
處理 14 queen
結果：[0, 2, 4, 6, 11, 9, 12, 3, 13, 8, 1, 5, 7, 10]
處理 15 queen
結果：[0, 2, 4, 1, 9, 11, 13, 3, 12, 8, 5, 14, 6, 10, 7]
處理 16 queen
結果：[0, 2, 4, 1, 12, 8, 13, 11, 14, 5, 15, 6, 3, 10, 7, 9]
處理 17 queen
結果：[0, 2, 4, 1, 7, 10, 14, 6, 15, 13, 16, 3, 5, 8, 11, 9, 12]
處理 18 queen
結果：[0, 2, 4, 1, 7, 14, 11, 15, 12, 16, 5, 17, 6, 3, 10, 8, 13, 9]
處理 19 queen
結果：[0, 2, 4, 1, 3, 8, 12, 14, 16, 18, 6, 15, 17, 10, 5, 7, 9, 11, 13]
處理 20 queen
結果：[0, 2, 4, 1, 3, 12, 14, 11, 17, 19, 16, 8, 15, 18, 7, 9, 6, 13, 5, 10]
處理 21 queen
結果：[0, 2, 4, 1, 3, 8, 10, 14, 20, 17, 19, 16, 18, 6, 11, 9, 7, 5, 13, 15, 12]
處理 22 queen
結果：[0, 2, 4, 1, 3, 9, 13, 16, 19, 12, 18, 21, 17, 7, 20, 11, 8, 5, 15, 6, 10,
14]
處理 23 queen
結果：[0, 2, 4, 1, 3, 8, 10, 12, 17, 19, 21, 18, 20, 9, 7, 5, 22, 6, 15, 11, 14,
16, 13]
處理 24 queen
結果：[0, 2, 4, 1, 3, 8, 10, 13, 17, 21, 18, 22, 19, 23, 9, 20, 5, 7, 11, 15, 1
2, 6, 16, 14]
處理 25 queen
結果：[0, 2, 4, 1, 3, 8, 10, 12, 14, 18, 20, 23, 19, 24, 22, 5, 7, 9, 6, 13, 15,
17, 11, 16, 21]
處理 26 queen
結果：[0, 2, 4, 1, 3, 8, 10, 12, 14, 20, 22, 24, 19, 21, 23, 25, 9, 6, 15, 11,
7, 5, 17, 13, 18, 16]
處理 27 queen
結果：[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 18, 22, 24, 26, 23, 25, 5, 9, 6, 15, 7,
11, 13, 20, 17, 19, 21]
處理 28 queen
結果：[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 22, 24, 21, 27, 25, 23, 26, 6, 11, 15,
17, 7, 9, 13, 19, 5, 20, 18]
處理 29 queen
結果：[0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 19, 23, 25, 20, 28, 26, 24, 27, 7, 11,
6, 15, 9, 16, 21, 13, 17, 22, 18]
處理 30 queen
```

用 backtrack 的方式，在 30 的時候就做不出來了！

有幾例使用基因演算法，如

1. Genetic algorithm vs. Backtracking: N-Queen Problem (https://towardsdatascience.com/genetic-algorithm-vs-backtracking-n-queen-problem-cdf38e15d73f)
2. An Adaptive Genetic Algorithm for Solving N-Queens Problem (https://arxiv.org/pdf/1802.02006.pdf)

In [ ]: