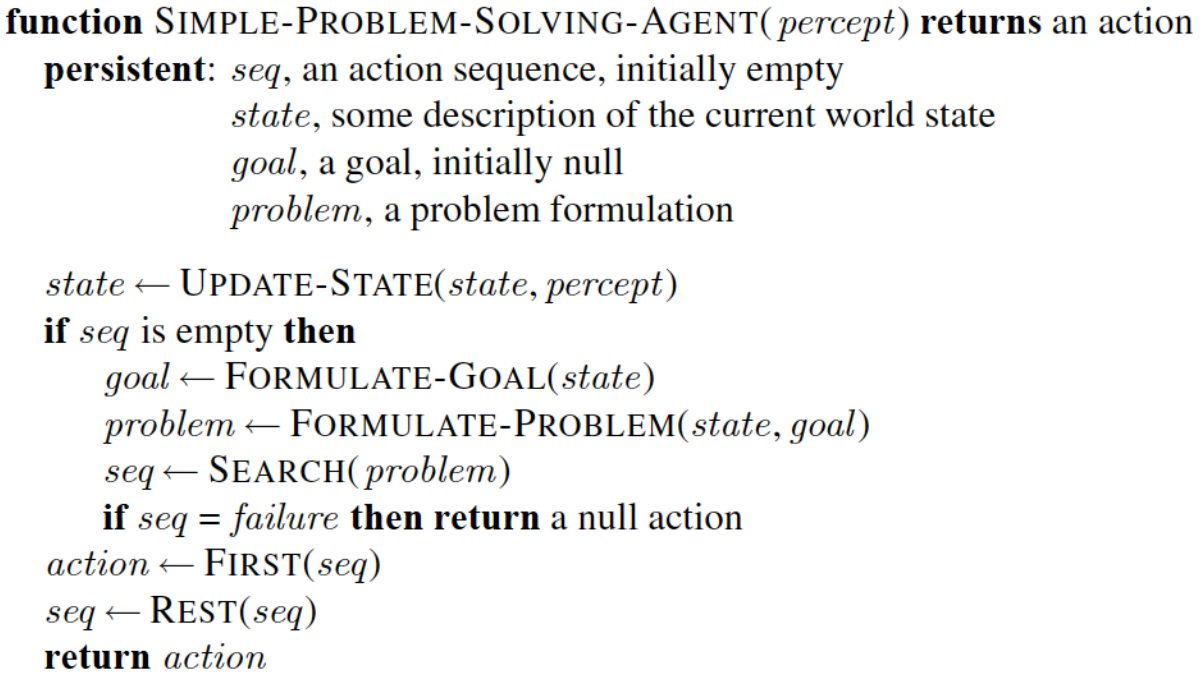
**Homework assignment#1 (Chap3)**

106971001 林上人

1. **Pseudo codes documentation**

Pages: 3



②

①

③

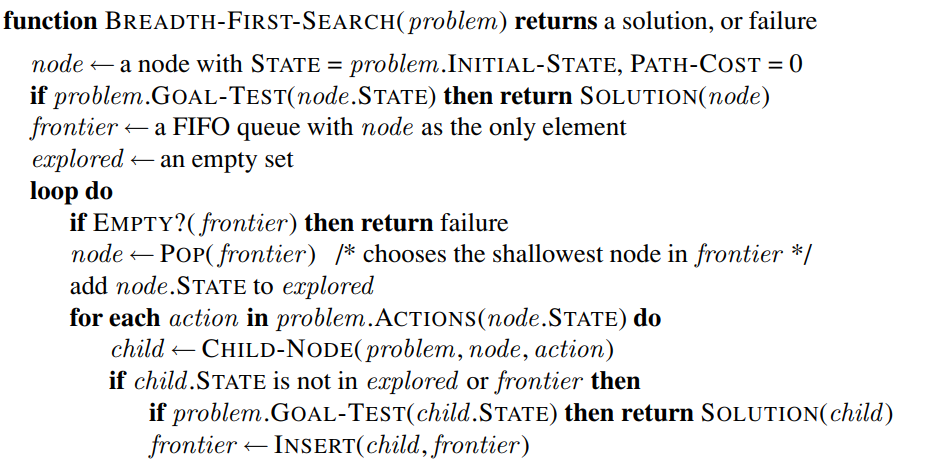
④

簡單的問題求解Agent設計

一開始persistent的部分先説明後並定義後面函數會用到的變數名詞，類似全域變數，包含seq用來存放動作序列，state為對目前的狀態描述，goal表示欲達成的目標，problem則代表問題的描述，意指在給定目標的情況下要考慮哪些狀態和動作的過程。

整個流程包含三個部分，分別為Formulate、Search、Execute，一開始①根據狀態和感知更新自己的狀態,後檢查動作序列是否為空，若不是空的表示目前其實是在一個解決問題的動作過程中,若爲空則進入②(Formulate)的階段,並制定目標和界定問題,然後在③(Search)尋找解決問題的動作序列,最後進入④(Execute)階段,從動作序列取出第一個動作後更新序列為剩餘的動作並將取出的動作回傳,function每次回傳一個動作,直到動作序列為空才會再次進入②(Formulate) 、③(Search)階段。

Pages: 47



③

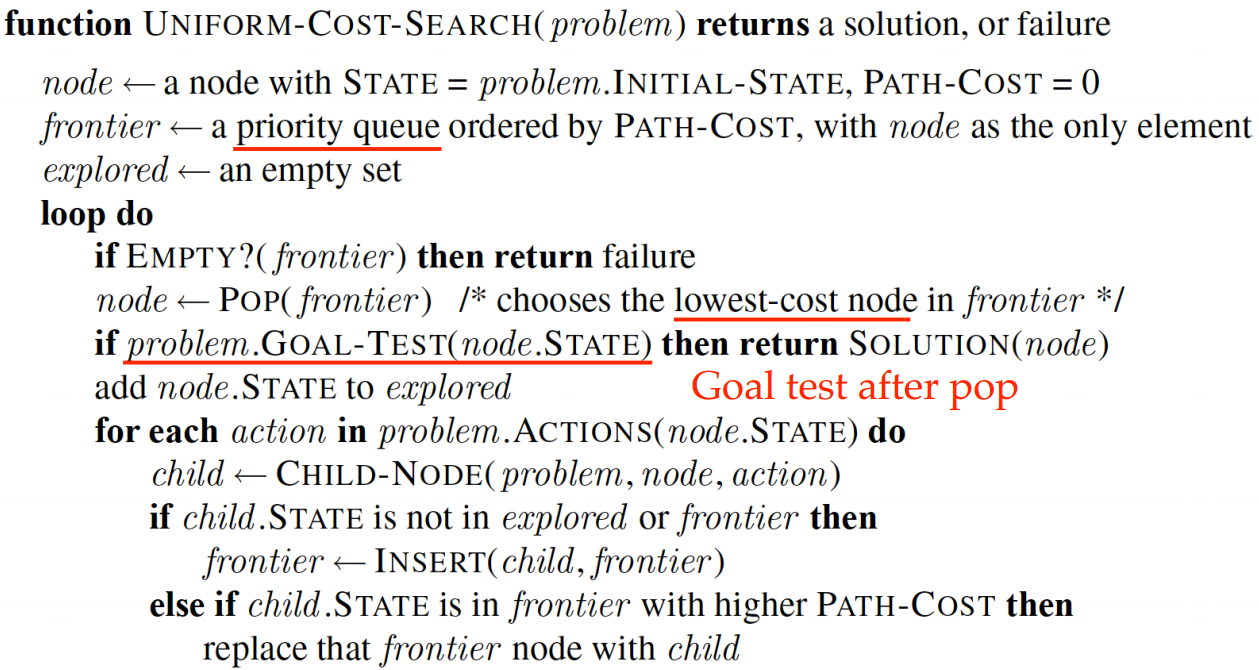
②

①

BFS演算法 - 先檢查最淺節點

此處使用BFS演算法來解決問題，①檢查問題的初始節點是否已經能夠通過GOAL-TEST,通過就回傳Solution，否則初始節點就為第一個frontier佇列元素，並利用FIFO的佇列結構實作frontier以達到BFS先檢查最淺節點的特性，explored集合用來存放已經遍歷過且不屬於frontier的節點,後續的步驟以迴圈執行， ②首先檢查frontier佇列内還有沒有元素，沒有則表示找不到解就回傳failure，否則就從frontier佇列中POP出一個節點，因爲放在佇列中的節點都是沒有通過GOAL-TEST，所以取出後也直接把它加入explored集合，然後③針對取出的節點找出子節點並檢查，如果子節點不屬於explored集合，也不屬於frontier佇列，就先對它做GOAL-TEST,若通過則表示找到解回傳Solution,沒通過就加入frontier佇列之中，整個迴圈會執行到frontier為空回傳failure（②）或child通過GOAL-TEST回傳Solution（③）爲止。

Pages: 61



③

②

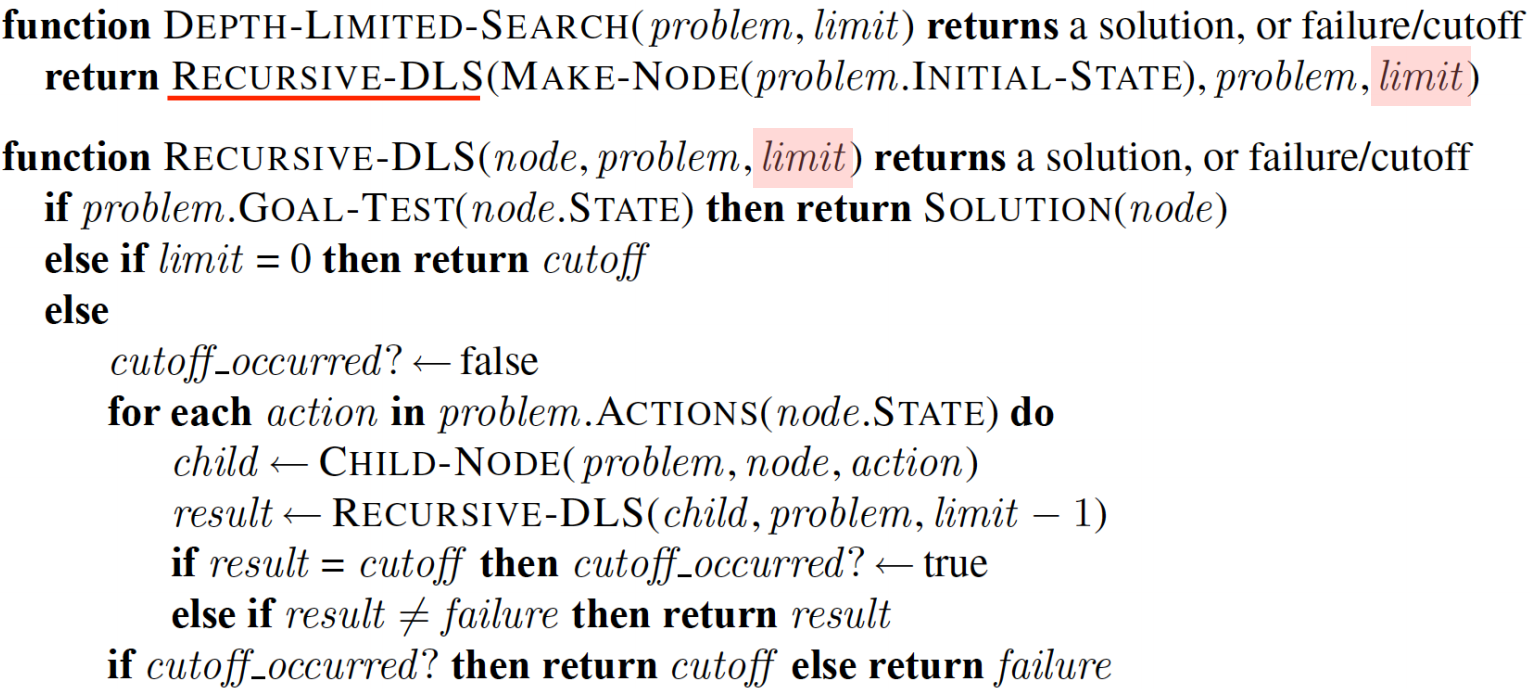
④

①

UCS演算法 – 改良BFS,考慮路徑成本

演算法類似BFS演算法，但①不檢查問題的初始節點而是初始節點直接就為第一個frontier佇列元素，並把frontier佇列改成以PATH-COST排列的優先權佇列，explored集合一樣存放走過的節點,後續的步驟以迴圈執行，②首先檢查frontier佇列内還有沒有元素，沒有則表示找不到解就回傳failure，否則就從frontier佇列中POP出PATH-COST最低的節點，對節點做GOAL-TEST，通過就回傳Solution，否則把它加入explored集合，然後③針對取出的節點找出子節點並檢查，如果子節點不屬於explored集合，也不屬於frontier佇列，就依PATH-COST加入frontier佇列之中，但是④如果子節點已經存在於frontier佇列中，就表示有其他路徑已經到達過此節點，此時就必須比較到達的PATH-COST，如果新的PATH-COST比較低就用這個子節點替換frontier佇列中的節點資料，因此即使目標節點已經先走到過也會在frontier佇列中不斷更新成最低成本的路徑，整個迴圈會因為無解而回傳failure或是執行到目標節點成為frontier佇列中成本最低的節點時通過POP和GOAL-TEST回傳Solution爲止（②）。

Pages: 85



④

⑤

③

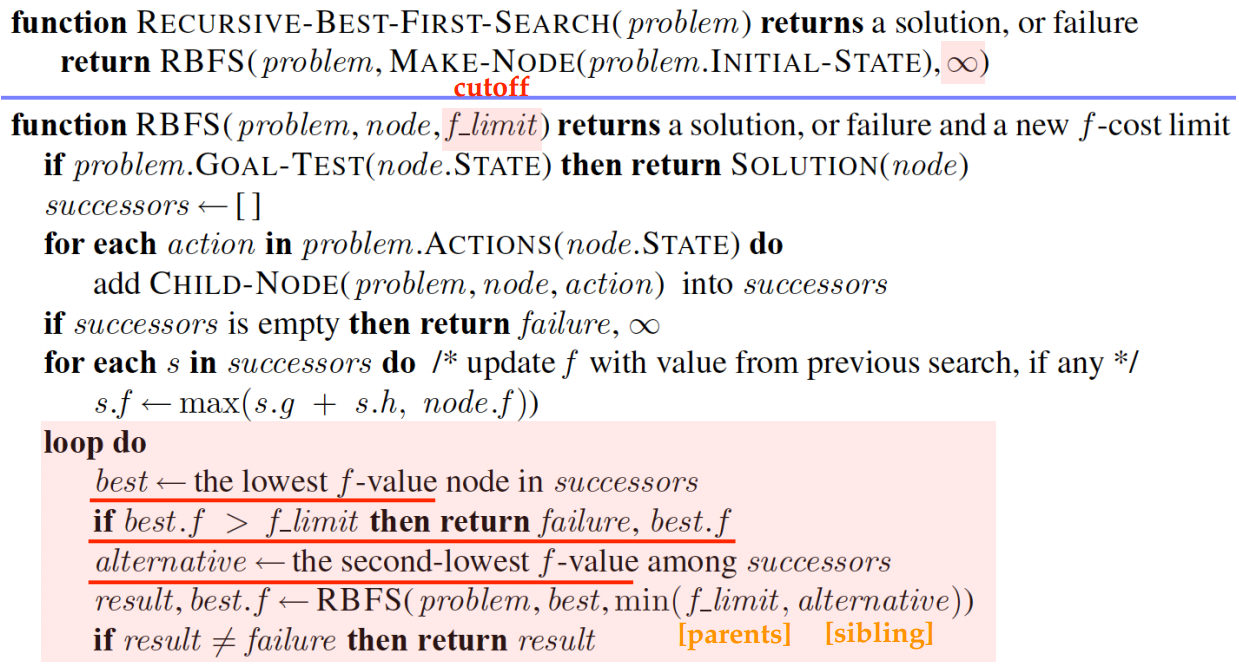
②

①

Depth limited search – 改良原DFS, 限制深度limit

此處以遞迴的方式實作Depth limited search，①Depth limited函式將初始節點和limit作為參數呼叫遞迴函式Recursive-DLS，而Recursive-DLS函式內則②先檢查傳入的參數節點是否可以通過GOAL-TEST，可以就回傳Solution，或③傳入的limit等於0則回傳cutoff，意指已經到達指定的深度，如果都不是則先宣告變數cutoff\_occurred等於false，然後④取出參數節點的子節點，並將子節點和深度-1作為參數呼叫自己，直到遞迴跑完可以取得result，因為Recursive-DLS有三種回傳值，所以result有三種可能分別為Solution、failure、cutoff，檢查result若為cutoff表示已經到達指定深度但是還未找到解，修改cutoff\_occurred值為true，如果result不是failure則代表是Solution，直接回傳result，最後⑤檢查cutoff\_occurred值若是true則回傳cutoff，若是還未達到指定深度但是找不到解就回傳false，這種可能就是子樹的規模比指定的深度小，而Depth limited函式的結果就和Recursive-DLS一樣分為找到解:Solution、未找到解但完成深度limit的搜尋:cutoff、未找到解且搜尋深度沒有達到limit:failure。

Pages: 137



⑤

④

③

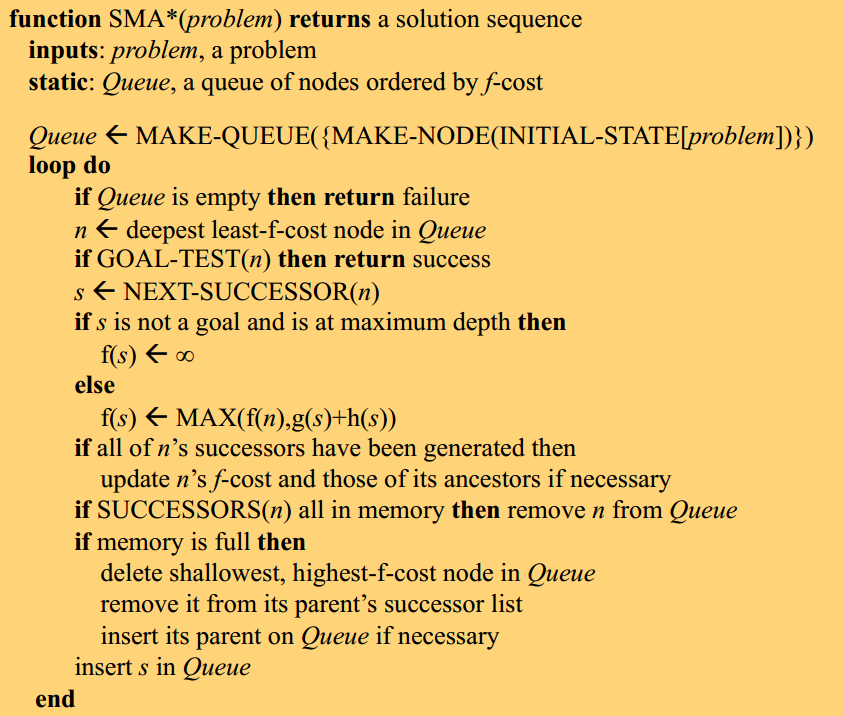
②

①

RBFS演算法 – linear space和遞迴的類standard best-first search

遞迴的方式實作best-first search，①以初始節點和作為參數呼叫遞迴函式RBFS，RBFS②首先檢查傳入的參數節點是否能通過GOAL-TEST，否則③創建一個successors陣列並放入所有的子節點，如果沒有子節點就表示現在這個節點已經沒有路可以走向目標節點，就回傳failure，接著④利用參數node更新successors的f值，意思是若此節點曾經被展開過，則節點的f-value應該要是successor的lower-bound，避免冗餘動作造成效率不佳的問題，所以取max(s.g + s.h, node.f)作為successor的f-value，但是如果這個節點沒有展開過則successor的f-value就等於自己的成本估計值( s.g + s.h )，之後以迴圈方式執行後續動作，⑤先取得f-value最低的子節點與f\_limit相比，因為f\_limit記錄的是祖先節點最好的估計代價，所以如果f\_limit比較小則表示有其他路較佳，因此僅回傳failure和此路徑知道的最小估計代價，儲存的successors陣列不保留以節省記憶體空間，如果f-value比較小則拓展這個子節點，以parents節點或sibling節點中較小的f-value作爲新的f\_limit參數呼叫遞迴函式RBFS，RBFS的回傳值有兩種可能，一個是Solution就直接回傳，另一個則是failure和一個數值，而這個數值即代表這個節點下的subtree的估計代價會大於等於這個數值，因此我們就以這個值替換原本的f-value,接著繼續跑迴圈直到找到目標節點。

Pages: 146



②

⑥

⑤

④

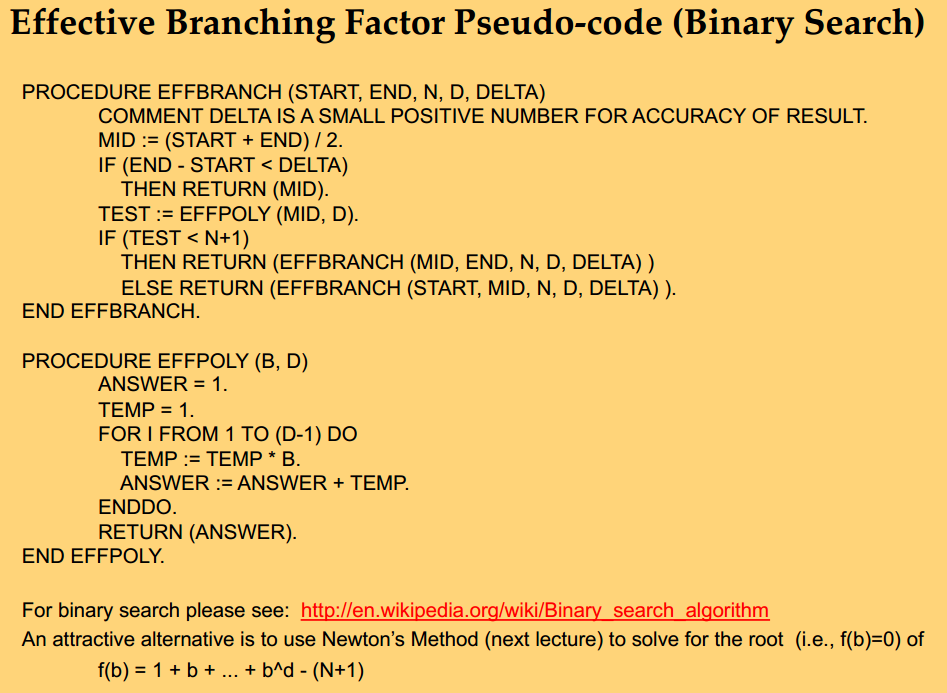
③

①

SMA\*演算法 – 改善記憶體使用情形的A\*演算法

定義一個f-cost的優先權佇列Queue並放入初始節點，整個流程以迴圈持續執行，①檢查Queue是否為空，是的話就回傳failure，然後②每次取出當前Queue中深度最深且f-cost最低的節點n，並對n做GOAL-TEST，若有通過就回傳success，若沒有③就繼續往下尋找n的子節點s，如果s不是目標節點而且已經達到搜尋的最大深度，我們就設定s的f-cost為∞，不然f-cost就是n的f-cost和g(s)+h(s)中較大的值，④如果所有n的子節點都已經出現過，就可以依據子節點更新n和他的祖先節點的f-cost到最小的f-cost，因為f(n)=g(n) + h(n)，其中g(n)的值可以被確實的算出來，⑤如果n的子節點都在記憶體中，我們可以暫時將n從Queue中移除，而⑥是SMA\*演算法中最重要的部分，如果記憶體滿了可以選擇將f-cost最高的節點移除，因為從當前情況看我們可視為從f-cost延伸的路徑都會有較高的f-cost，藉此解省記憶體的使用，但parent必須紀錄被丟棄的部分中最低的f-cost，以便最後仍然需要復原刪除的部分時可以使用，最後將s加入Queue中，持續直到找到目標節點為止。

Pages: 154



③

②

①

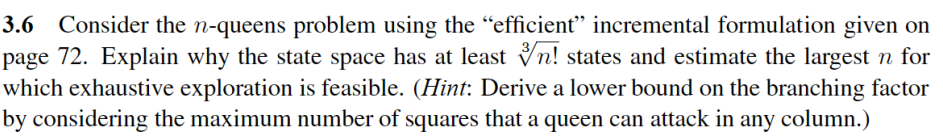
使用Binary Search的方式計算Effective Branching Factor

首先須知道Effective Branching Factor( 的定義，當expanded的節點總數為N，解的深度為d時要符合

N + 1 = 1 + +

此處使用binary search的方式計算，先估計一個的值區間可能介於START和END之間，然後①取START和END的中間值作為的猜測值，並丟入EFFPOLY計算，③利用猜測值和d嘗試計算ANSWER = +1 = 1 + +，並回傳計算結果，最後②把猜測值的計算結果+1與真實N+1做比較，若小於N+1表示低了，因此把MID到END作為新的區間再猜測新的值做計算，若大於N+1表示高了，因此把START到MID作為新的區間再猜測新的值做計算，持續執行到算出正確的。

1. **Exercises**

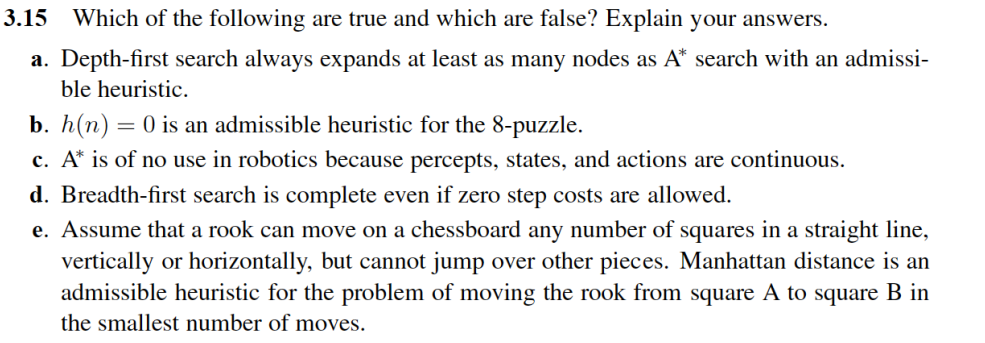


Ans :

因皇后放置後最多可攻擊三格，所以當第一欄有n個位置可以放時，第二欄最少還會有n-3個位置可以放，依此類推可寫成

State space 至少有

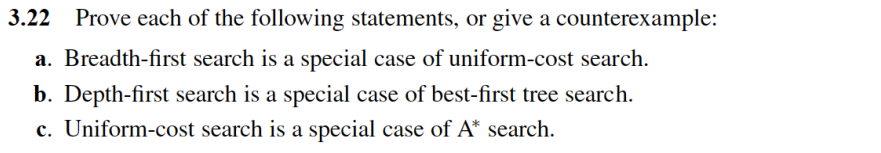
-------------------------------------------------------------------------------------------------------



Ans :

1. False : DFS不考慮成本會一直往下搜尋，而A\*還考慮成本因此會拓展f低的路徑，若目標剛好在DFS不用回朔的路徑，拓展的路徑就可能會比A\*少。
2. True : 因為0已經是最小了，不可能高估。
3. False : 連續動作還是可以被離散化或概略化，所以還是可以使用。
4. True : 因為BFS不考慮成本，只要是finite就是complete
5. False : 因為在直線上城堡在一步內移動的距離是不限格數的，所以Manhattan distance 是有可能高估的

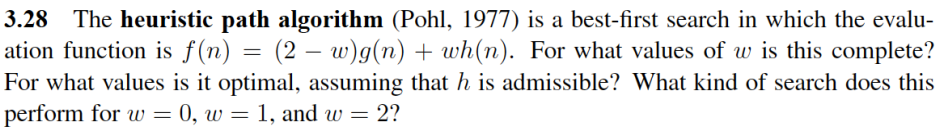
-------------------------------------------------------------------------------------------------------



Ans :

1. True : uniform-cost search就是從BFS考慮路徑成本演變而來，所以當cost都相等時uniform-cost search就會表現的與BFS一樣
2. True : 當定義best-first tree search的heuristic function定義為 f(n) = -depth(n)就可以表現得和DFS一樣
3. True : A\*的heuristic function : f(n) = g(n) + h(n) ，當定義h(n)=0時就會等於Uniform-cost的heuristic function，他們的表現就會一樣

-------------------------------------------------------------------------------------------------------



Ans:

Uniform-cost search和A\*是complete, Greedy不是complete,

如果要符合complete → 0 w < 2

Uniform-cost search和A\*是optimal,且h不能高估

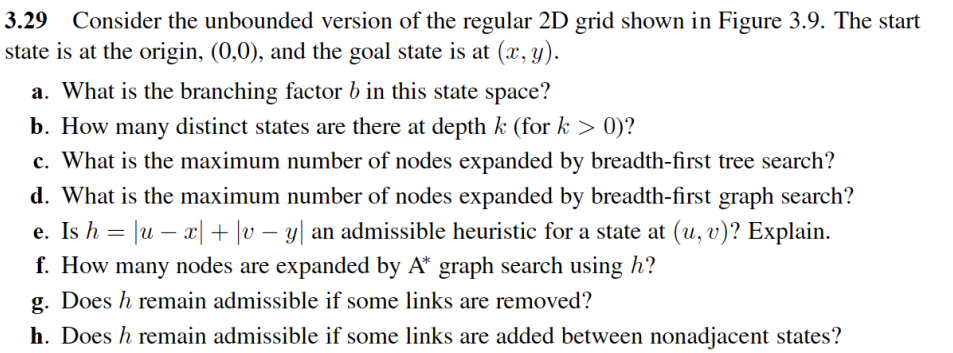
如果要符合optimal → 0 w 1

w = 0 , f(n) = 2g(n) → Uniform-cost search

w = 1 , f(n) = g(n) + h(n) → A\* search

w = 2 , f(n) = 2h(n) → Greedy best fisrt search

-------------------------------------------------------------------------------------------------------



Ans:

a.因為是無邊界，所以每個點都可以向四方發展，分支因子為 4

b. dinstinct states，一個點就是一個state，因為是at不是when所以我們計算at depth = k的node有4k個，所以states有4k

c.因為樹的每一個node都有4個子節點，因此最差探索點數量發生在所有子節點都走一遍，假設goal node在depth=k上，則

節點數量n =

=

又因2D grid產生是正方形，所以可知 k = |x|+|y|

→ nodes =

d. 因為是graph，走過的點就不走，因此最差探索點數量發生在所有節點走一遍，假設goal node在depth=k上，則

節點數量n = 1 + 4(1+2+3+4….k)

= 1 + 2(k+1)k

又因2D grid產生是正方形，所以可知 k = |x|+|y|

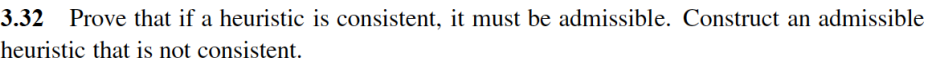
→ nodes = 1 + 2( |x| + |y| + 1)(|x| + |y|)

e.符合admissible，因為每格邊距都是固定1，所以不會高估

f. |x|\*|y|個節點，因為使用*h*時在origin(0,0)到目標(x,y)之間的路線都是最佳解

g.是，因為移除links可能讓最佳路線變長，但是用*h*估算路線還是不會高估所以符合admissible

h.不會，因為加入新的節點在不相鄰的點之間可能讓最佳路線變短，因為*h*估算路線不包含對角線之類的所以可能高估，不符合admissible



Ans:

admissible : ≦ k(n)

consistent : ≦ c(n, a, ) +

Prove :

1. 令k(n)表示從n到目標的實際最低成本
2. 令c(n, a, ) 表示從節點n經過動作a到達的成本
3. 令h(n) 表示從n到目標的估計成本
4. 根據定義，若是consistent就會符合 h(n) ≦ c(n, a, ) +
5. 假設n為Goal，則 = 0 ≦ k(n) 成立
6. 假設n不為Goal距離Goal k步 ,且為Goal,則從n到 寫作

h(n) ≦ c(n, a, ) +

∵ (5)成立

∴ h(n) ≦ c(n, a, ) + = k(n) 成立

可知若heuristic是consistent則他必定是admissible

1

A

定義

k(x) = m

h(x) = 2 x (m/2)

h(A) = 4 ≦ k(A)

h(B) = 2 ≦ k(B) = 3

h(G) = 0 ≦ k(G)

∴ admissible

h(A) = 4 > 1 + h(B) = 3

∴ not consistent

3

G

B