人工智慧-hw02 標靶治療

40823117L 資工系 方國丞

1.詳細說明你所使用之機器軟硬體規格及作業系統、開發軟體版本、如何執行程式相關資訊。另外請提供你的連絡電話,以便不時之需。

```
機器：MacBook Pro 2023
CPU/GPU: M2
Python version: Python 3.10.9
```

如何執行程式

1. 使用 input_generator 來生成五個 input file `python input_generator.py`
2. 執行 IDS `python IDS.py`
3. 執行 IDASTAR `python IDASTAR.py`
4. 輸出結果分別為output-IDS.txt與output-IDASTAR.txt

2. 首先你先設法製作測試用的輸入檔至少5個。盤面尺寸需有大有小,解題難度也不同,由你決定。請說明你如何製作這些測試用的輸入檔。(提示:可考慮用亂數產生)

input_generator 使用 random.randint(1, 10) 生成一個介於 1 和 10 之間的整數，當作棋盤的大小。再使用 random.uniform(0.3, 0.7) 生成一個介於 0.3 和 0.7 之間的cancer_ratio，作為癌細胞在這個案例中的比例

generate_input_file函式這是實作如何生成數值(0 or 1)，random.random() 生成一個介於 0 和 1 之間的隨機浮點數，如果這個數小於 cancer_ratio，則設為 1，否則設為 0。

3. 兩支程式之原始碼中應加註解,請說明如何執行這兩支程式。

- 執行IDS演算法 `python IDS.py`
- 執行IDASTAR演算法 `python IDASTAR.py`

4. 請說明第一支程式IDS你使用什麼方法、甚麼資料結構、什麼技術(操練要項)來解決這個問題,並請說明你測試一些盤面時的表現如何、耗用的時間及空間為何(假設盤面有n個位置)、你的程式能解到多大盤面的題目?請你用一些例子輔助說明。(解題愈快、說明愈清楚的程式成績愈高。)

IDS的那支程式實做了深度優先搜尋(DFS)和廣度優先搜尋(BFS)

在程式中使用的資料結構和技術：

1. 使用list來表示棋盤狀態和移動。
2. 使用recursive執行具有有限深度的DFS。
3. 使用loop實現迭代加深，逐步增加搜尋深度。

4. 程式的效能取決於輸入棋盤的大小和複雜度。對於較小的棋盤和較少的癌細胞，該演算法可以快速找到最優解。然而，隨著棋盤大小和癌細胞數量的增加，搜尋空間呈指數級增長，有可能會無法找到最佳解

時間和空間複雜度：

時間複雜度：IDS演算法的時間複雜度為O(n^m)，其中n是branching factor，m是所求解的深度 空間複雜度：IDS的空間複雜度為O(n*m)，比BFS好但比DFS差

使用不同大小的輸入棋盤進行測試時，程式的效能會有所不同。例如： 較小的棋盤和較少的癌細胞： 程式可以快速找到解決方案，並佔用較少的記憶體。 更大的棋盤和更多的癌細胞： 程式尋找解決方案所需的時間較長，佔用較多的記憶體，因為搜尋空間更大。

舉例來說8單位長度的input `1 1 0 0 1 1 0 0` 在IDS的執行時間為 2.773651 secs

而假如把其中1個0換成1 `1 1 0 0 1 1 0 1` 執行時間就會變成 3.247244 secs

而假如把其中1個1換成0 `1 1 0 0 0 1 0 0` 執行時間就會變成 0.001000 secs

多一個1變成9單位 `1 1 0 0 1 1 0 0 1`

執行時間就會變成 43.24183 secs

而假如是多個0，一樣是9單位則會 `1 1 0 0 1 1 0 0 0` 執行時間就會變成 532.19231 secs

結論:

與input的長度跟0與1的比例有關，長度越長的input所需時間越高，而細胞好壞比例只要約接近一半一半則所需時間會越高，而當靠近一半一半的輸入時，越多1則會需要更長的時間來搜索

較小的棋盤和比例較一面倒的輸入： 程式可以快速找到解決方案，並佔用較少的記憶體。 更大的棋盤和比例平均的輸入： 程式尋找解決方案所需的時間較長

能夠解的盤面長度: <8~9

5. 請說明第二支程式IDA*你使用什麼方法、甚麼資料結構、 什麼技術(操練要項)來解決這個問題,並請說明你測試一些盤面時的表現如何、耗用的時間及空間為何(假設盤面有n個位置)、你的程式能解到多大盤面的題目? 請你用一些例子輔助說明。(解題愈快、說明愈清楚的程式成績愈高。)

IDA是一種啟發式搜索演算法，結合了迭代加深深度優先搜索（IDS）和A搜索的優點

程式中使用的資料結構和技術：

1. 使用 list 來儲存棋盤狀態和移動。
2. 使用 recursive 執行具有有限深度的深度優先搜索。
3. 使用一個 heuristic function 來估計到達目標的成本。
4. 使用一個set來存儲訪問過的狀態，以防止重複訪問相同的狀態。

程式的效能取決於輸入棋盤的大小和複雜度。與IDS不同是使用heuristic function可以讓演算法先探索更有希望的分支

時間和空間複雜度：

時間複雜度：IDA算法在worse case下具有指數時間複雜度與IDS相近，但對於具有良好啟發式函數的問題，通常比IDS更快。 空間複雜度：IDA空間複雜度為O(n)，n是解的深度，因為只在記憶體中存儲當前路徑。

當使用不同大小的棋盤和不同的輸入進行測試時，程式的效能會有所不同。例如：

舉例來說15單位長度的input 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 在IDA*的執行時間為 36.425255 secs

而假如把其中1個0換成1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 執行時間就會變成 19.315736 secs

而假如把其中1個1換成0 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 執行時間就會變成 87.206093 secs

多一個1變成16單位 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1

執行時間就會變成 105.079161 secs

而假如是多個0，一樣是16單位則會 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 執行時間就會變成 166.447105 secs

結論:

與input的長度跟0與1的比例有關，長度越長的input所需時間越高，而細胞好壞比例只要約接近一半一半則所需時間會越高

較小的棋盤和比例較一面倒的輸入： 程式可以快速找到解決方案，並佔用較少的記憶體。 更大的棋盤和比例平均的輸入： 程式尋找解決方案所需的時間較長，但由於啟發式函數引導搜索，仍然比IDS效率更高。

能夠解的盤面長度: <20

6. 請說明你做此作業所碰到的一些狀況及困難。

在IDS演算法中，input太大會導致執行時間過久，也不確定是不是演算法出了問題，因為在比較少input的狀況下輸出都是正確的，優化時間複雜度有點困難 IDA*則是在比較複雜的input也會遇到這個問題，但是能接受的input量比IDS多很多

7. 請列出你的參考文獻(含網站)來源,並請說明參考了那些部份用於作業中。

me and gpt-4 for writting part