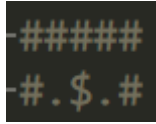


# Theory of Computer Games (Fall 2018)

## Homework #1

資工所碩二 劉又萱 R06922041

- Implementation
  - How to **compile and run**
    - ◆ 編譯:我提供了一個 Makefile，可以透過 **make** 指令來編譯，並生成 r06922041 的可執行檔。
    - ◆ 執行:**./r06922041 < ../testdata/XXX.in**
  - What algorithm and heuristic you implement
    - ◆ **不重複性:**利用 unordered map，將每個走過的盤面編碼成唯一的識別碼
    - ◆ **剪枝:**因為實際上倉庫番實在是太容易某個箱子被卡住(deadlock)，然後我們還繼續往下測試，直到其他箱子都推到目標，我們才會發現這裡是死路。這樣實在是太浪費時間與空間了，因此找到好的剪枝方法是很重要的。如此一來可以先判定沒有 deadlock，再放入序列裡面等待下次 Pop 出來。
      - Simple deadlocks:
        - 利用 goal 拉箱子到各個地點，紀錄每個目標點能夠停留箱子的位置集合。
        - 初步判斷是否有箱子再沒有任何 goal 到的了的地方，有就是 deadlock
        - 看是否有一個 goal 上的合法區域沒有任何箱子，沒有箱子就是 deadlock
      - Freeze deadlocks:
        1. 有時候有箱子推到被卡死的位置，永遠都無法再被推
        2. 利用遞迴方式判斷箱子垂直跟水平是否被卡死
        3. 如果這些被卡死的箱子有任何一個還沒推到 goal，就是 deadlock
      - Corner deadlocks: (Freeze deadlocks 包含此)
        1. 判斷是否箱子的相鄰兩邊是牆壁，如果是就是 deadlock
      - $\sqcap$  deadlock:
        1. 如果有像是這種箱子被包圍的情況，而且旁邊這兩個空位沒有 goal，就是 deadlock



2. 四種旋轉方式

■ 2\*2 deadlock: (Freeze deadlocks 包含此)



1.

2. 如果兩個箱子相鄰，而且又遇到牆壁相鄰，或是交叉的話都算 deadlock

### ◆ 實際測試:

- BFS(Breath-first Search): 採廣度優先，而且確保一定可以得到最佳解，但是因為作業有限制記憶體使用量，所以如果單純使用 BFS，則會超過 4G 使用量。
- DFS(Depth-first Search): 採深度優先，但是很容易會陷入很深的死結節點(deadlock state)，而且時間通常很久，small.in 第八個以後都會跑好幾分鐘。
- DFID: 由於一開始決定的 cut off depth 是 heuristic，如果沒選好，可能就會重新跑一次 traverse tree，這樣反而會浪費時間。像是一開始想說設定深度 100，但是遇到最終盤面其實是第 101 層的是後就反而很浪費時間。(雖然老師說過，當廣度越廣的時候，其實多花的成本只有 7%，但是對於這次的側資而言，經過測試，我覺得不太適合)
- With heuristic
  - 成本 =  $g + h$ (Heuristic function)。g 表示已經用掉的成本，也就是走到這個狀態所需的步數；h 採用 Manhattan 來求得每個箱子到最近目標的距離和+玩家距離最近箱子的 Manhattan 距離
  - A\*: 逐漸加深深度優先搜尋+最佳優先搜尋。相較上面測試過的方法是比較快的。但是有時候走到越深，廣度過於廣，找答案要找很久，像是 large.in 要 3 分鐘，large2. 要 5 分半。因此我後來改用下面的方法。
  - **雙向 A\*: (正向用 A\*，反向用 BFS，開 thread)**
    - ◆ 正向: 與 A\* 相同，會一邊找，一邊確認是否反向已經遇到該狀態，如果遇到就算是找到路徑(雖然不能保證是最佳解，但是速度快很多)
    - ◆ 反向: 我會先把所有箱子擺到 goal 的位置，並把人放到棋盤上，每一個方向可以選擇拉與不拉兩種狀況往下延伸

- Experiment

	DFS	DFID	BFS	A*	雙向 A*
Small	3m3.65s	2m13s	0m9.89s	0m0.565s	0m0.165s
Large	8m8.44s	4m35s	3m40s	2m3.752s	0m2.32s
Large2	12m32.56s	13m29.065s	8m2.0s	5m24.857s	1m54.139s

- DFS (光是 large2.in 跑 10 分鐘都跑不完)

- ◆ 因為畢竟不是每種 deadlock 我都有定義，像是 Closed diagonal deadlocks、Corral deadlocks、Deadlocks due to frozen boxes 這三種因為如果要定義的話會花更多成本，我就沒有偵測。那由於不是每種 deadlock 都有被我偵測出來，所以如果節點其實已經是上述某種 deadlock，程式還繼續 explore 它，那就會徒勞無功，跑太久。

- BFS VS 單向 A\*剪枝(with large.in)

real	3m40.775s	VS	real	2m5.752s
user	5m21.203s		user	2m2.311s
sys	0m25.109s		sys	0m2.508s

A\*透過 heuristic+g 來拿節點運算，讓較有可能的節點先探索，所以也會快很多。

- 單向 A\*剪枝 VS 雙向 A\*剪枝(with large.in)

real	2m5.752s	VS	real	0m2.320s
user	2m2.311s		user	0m4.067s
sys	0m2.508s		sys	0m0.230s

因為有時候順向有時候會繞路，如果透過平行的雙向運算，那就能夠更快找到解答，因為兩邊同時搜尋，一定至少減少一半的時間。

- DFID VS 單向 A\* (with large2.in)

real	13m28.965s	VS	real	5m24.857s
user	13m0.375s		user	5m18.874s
sys	0m14.391s		sys	0m2.861s

層數是設定 100 每次+20

因為設定層數也是關鍵的一部份，好的層數限制以及好的每次增加次數，才能讓程式更快，但是我測試後這些側資用了 DFID 並沒有比較好。因為 e 不夠大，只有 4，所以逐漸加深演算法並沒有太大的幫助。

- 單向 A\*剪枝 VS 雙向 A\*剪枝(with large2.in)

real	5m24.857s	VS	real	1m54.139s
user	5m18.874s		user	3m25.328s
sys	0m2.861s		sys	0m15.375s

因為有時候順向有時候會繞路，如果透過平行的雙向運算，那就能夠更快找到解答，因為兩邊同時搜尋，一定至少減少一半的時間。

- 雙向 A\*不減枝 VS 雙向 A\*剪枝(with large.in)

real	0m3.832s	VS	real	0m2.320s
user	0m5.906s		user	0m4.067s
sys	0m1.297s		sys	0m0.230s

剪枝真的很重要，幫我省下許多白計算的時間。

- 雙向的效果：
  - ◆ 開 thread 後，可看出反向找到的序列其實大於最後找到的序列的一半，成效很好。不過先搜到的結果不一定是最佳解

```
tinaliu@Acer: /mnt/d/TINA/sokoban
tinaliu@Acer:/mnt/d/TINA/sokoban$ time ./r06922041 <./large2.in
-----Find BackPath: 229
ddrrruulruurlddddlluurldddrruuruulldldddrruulruulldrrruurllrddrudlluur
urddrdlluurdlillurdrdrddldlluuuuddrruulruurduulllldrlddddrruuruulldld
ddrluururddldlluurldddrruulldrrddlddrulruurldlduurdd
325
ddrruulduurullldurdlrddldurddluurullldrlulldrrrddldlluurldldddrdlluu
urruulldddlddrurullruurldldlluurldddrruulldlddrurulluulldrrruur
uillrddrdlluurdrddluurdlillurdrdrddldlluuuuddrruulruurduulllldrlddd
ddrruuruulldlddrulruurddldlluurldddrruulldrrddlddrulruurldlduu
rdrdd
Return-----
-----Find BackPath: 140
ulldldrdddrulruulldurdrddllurruulldruuldrulldddruulruurdlilruur
rdldlddrddrdllulururddldldlurruuldrddruulrddlddruuu
182
drdrduuulldurdlilruurldduulldrruulldldrdddrulruulldurdrddllurrr
uulldruuldrullddddrduulruurdlilruurldlddrddldlulururddldldlurruul
drdruuulrddlddruuu
Return-----
-----Find BackPath: 126
drurrdldlulruurduurddldurddldldlddrurdluurdruulldruulldlddrurrdld
lulruururddldldlddrurdluurdrrldldlu
171
dlulruulldlddrurrdldlulruurddldldlddrurrdldlulruurduurddldurddldl
ldlddrurdluurdruulldruulldlddrurrdldlulruurddldldlddrurdluurd
ruldrrldlu
Return-----
-----Find BackPath: 120
uurrdldlruurldlddrurldrddldurdlulldrrldruurduulldldlruurrd
ldlruulldldrrulldldruulldrrldr
202
rrdrddldlulldurdrddruulldlruulldrrddldllurldldlurdrddruulldldldld
ruurrdldldlruulldlddrurldrddldurdlulldrrldruurduulldldlruurrd
ddldlruulldldrrulldldruulldrrldr
Return-----
-----Find BackPath: 126
rdrulldlurdrddldlululruurdddrduulldldlruurdrurrdldddldlruuul
ldlddrulldlruurduurddlruulldlddr
166
ruulldlrdldlulruulldlruullddrulldrruulldlurdrddldlululruurlddddr
uuulldldlruurdrurrdldldlruuulldlddrulldruurduurddlruulldrr
ddruu
Return-----
```

- 最好的結果：
  - ◆ Small.in 0.165s

```

tinaliu@Acer:/mnt/d/TINA/sokoban/source$ time ./r06922041 <../small.in
5
urrdl
10
rddlddruuu
45
rdddluruu11ld1lurrrurddd1luru11ld1rrrrurd
48
rrdruldlu11ld1lurrrrrrdrul11rrruulldurddl
61
1111ulldrr1111ld1luurrrrrurd1111ulddurrrrrrrurd111111ulld
52
rddddd1lurduuuulldrdldluuulurdrdd1lurlddddruuul1u
62
rrdrulullddluurdrdd1ul1uurrdul11ddrruldddudrull1uurd
25
drdlruruldululddurrurd
107
ddrrruulrruulldurddd111lulurdldrrruulld1druurruulld1drdd1udlurruurddd11u
urlddruuulld1ldrruulldrr
184
uulld1lurrlld1luurdrddd1ludruuuruurddd1lurrd1luruulld1lur11ddrddruulurur
rurddd1luruul1111ddrddruuldrurrd11luruul11ddrulurruurddul1111ddrddrurr
ruld111ulurrrurddd11
real    0m0.165s
user    0m0.219s
sys     0m0.063s

```

### ◆ Large.in 2.32s

```

r06922041@linux10 [~/ComputerGame/hw1_public/r06922041] time ./r06922041 <../testdata/large.in
16
dddrulruldllu
188
drurrlld1lurdrdd1ludruuullddruuuldrdd1lururdrruldrulld1l1dddrduud1luuulurdddrulld1l1dddrduulurrd1r
urdd1lruulldullddrddruud1luur1luurdrulld1l1dddrduud1luuulurdddrulld1l1dddrduulurrd1r
48
rur1luurd1drurrd1l1uurd1drdd1lurruurdd1lru
99
luurrrdrdd1luruulld1ddrdrdd1lurldruulld1l1dddruldrurduulld1l1luruldrddrulld1u
45
rruuruuldduuldrurdd1ulld1ddrrulld1l1u
34
rurdddddruuullddrdd1ld1lur1u
32
lduuldrurrd1ld1lulurulrddd
31
ddrrrd1ldruulld1lurulurduud
51
dddrulld1ddruurrrulrdd1ulld1lurruuldrdd1lurruu
78
ruulurduurdd1ullddrddrrruuulld1luldrdrurruuldrdd1l1lurrrrd1lruuullddur
real    0m2.320s
user    0m4.067s
sys     0m0.230s

```

```

r06922041@linux10 [~/ComputerGame/hw1_public/r06922041] ../sokoban/verifier -i .
./testdata/large.in -o large.out
Stage #1: Accepted: 16
Stage #2: Accepted: 188
Stage #3: Accepted: 48
Stage #4: Accepted: 97
Stage #5: Accepted: 45
Stage #6: Accepted: 34
Stage #7: Accepted: 32
Stage #8: Accepted: 33
Stage #9: Accepted: 49
Stage #10: Accepted: 76

```

### ◆ Large2.in 1m 54s

```

tinaliu@Acer:/mnt/d/TINA/sokoban$ time ./r06922041 <./large2.in
323
ddrrruulduuulldurrdlrrddlurdddluuuuulldrlulldurrrdddldl luulurrlddddrrddlluu
rrruuldddlddruruluurdudddlluuuldddrruuuulldlddddrruululullddddrruul
lrrddrudllluurddrdluurdlldlurdrddrdldl luuuudddrurruulurddulululldrrldddd
rruuuulldlddrlluuurddldldl luulurdddddrruululldrrddlddruluuuulldluurd
rdd
182
drdrduuddl lurdl lluruurdllduullddruruulldlddddrruulruuldlurdrddllldrrr
uulldldruuldrurrdlddddrruulurdl lluruurdllddrddrdldlulururddlulldruuldr
drurldldlddrddruuu
171
dlululldlddrururrdldlul luurduurddlulldlddrururrdldlul luurduuurrddlurddlul
ldrlddrurdl luurdruuulldruulldlddrururrdldlul luurduurddlulldlddrurdl luurdr
rrldluruul
200
rrdrdddlul luddlurrrdruuulldl ruuuldrdrddllluurdlldlurdrdrurllrruulul
lddddldruudrrulrrdlruuulldldrrulldl luurlddrurrdlluuuulldldlddrurrdlruu
uruulldlddrllururrdldlddruuulldrrlldr
162
ruuuldrddldlulldrrdrurruulldlurldlluurluurlddddrruuldrurruulldrdddl
ulldruulurdrurrdldrddllurruulldldldrrulldruulurddurrrddlurulldrrddr
uu
162
urdrululuuurrdldlrrldluddlrdurulldruuulullddddruudrruuulldrrddll
dluudrruullddrddldluruuldddldlurdruuulldruudrrulldruudruulldlurrrd
ru
145
luulldl luurdrddrdldlurllluurrdulldlullddddlluurldrruudrrurruululldrrldd
lluurldrruulurrdlddrdrurruulruuulduulldrrlldldldrrrlludrr
73
ddruudrruuldruuulldldldrrldruulldlrruuuldduuurrdlrrddllluulld
159
llllrruuullddrdrurruulul luurdrdluuurddlddrulduurduulldldrrddluruuulldl
ddrdrurldl luuluuurdrdduullddrurululdrurldldldruuuddrruulruulldrddruu
166
lurdruuuulldduuurrdddddllul luurllldrruldrdruuuulldduurddldruulurrrdd
ddldululldruuulldrddlurrrdruuuulldduurddlluuuurdurrrddldl luuuddrruuuulld
llddr
real    1m54.139s
user    3m25.328s
sys     0m15.375s

```

## ● Discussion

### ■ The complexity of a Sokoban puzzle.

◆ PSPACE-complete.

◆ State complexity: (m 是盤面的長度，n 是盤面的寬度)

● 粗估為  $\frac{mn}{8} * 5^{mn}$ ，以 7\*7 的棋盤為例，約  $10^{35}$

●  $5^{mn}$ : 先不考慮 player 的位置的話，有五種可能，分別是 #\$. -，所以每個位置都有五種方法。

●  $\frac{mn}{8}$ : 因為盤面具有對稱性，所以除以八

● 如果詳細來說，是  $\frac{mn}{8} \sum_{a=1}^{\frac{mn}{2}} C_{2a}^{mn} \frac{2a!}{a!a!} * 3^{mn-2a}$ ，以 7\*7 的棋盤為例，約  $10^{34}$

● a 是 \$ 的個數，a 也是 ' .' 的個數，所有先擺完這兩個的

話，共有  $\sum_{a=1}^{\frac{mn}{2}} C_{2a}^{mn} \frac{2a!}{a!a!}$

●  $3^{mn-2a}$  這個部分是五種減去 \$.，的所有排列

●  $\frac{mn}{8}$ : 因為盤面具有對稱性，所以除以八。但其實前面這



五種放的地方已經不能擺，所以其實要扣除

- ◆ Game Tree size: 給定一個盤面後，求所有合理的盤面數，a 假設給定的箱子數量，b 為給定的牆壁

- $C_a^{mn-b} * (mn - a - b)$ ，因為從全部的位置(mn-b)裡選 a 個地方把箱子，而最後 player 有(mn - a - b)位置可以擺

- The complexity of each algorithm.

- ◆ 定義符號:

- 節點分支度 b: 所有狀態節點的節點平均或最大分支度。由於我有確認不重複，因此 b 為不走到相同盤面的數目。

- 邊分支度 e:

- 由某狀態節點所衍生出可能重複的新狀態節點，其個數為邊分支度，可容許不完全相異的子節點
- 對於此遊戲，因為每次有上下左右四種方向可選，故 e 為 4

- 目標狀態的深度 d: 起始狀態節點到目標狀態節點的最短距離

- ◆ BFS(Breath-first Search)

- 時間複雜度  $O(b^{d-1} * 4)$ 
  - => 但因為我有 close list，所以 e 可以被 d 取代，為  $O(b^{d-1} * b) = O(b^d)$

- 空間複雜度  $O(b^d)$

- ◆ DFS(Depth-first Search)

- 時間複雜度  $O(4^D)$  (D 是因為可能不是最佳解)
- 空間複雜度  $O(D)$

- ◆ DFID

- 時間複雜度  $O(4^d)$
- 空間複雜度  $O(d)$

- ◆ A\*

- 時間複雜度: 最糟跟 DFS 一樣  $O(4^d)$
- 空間複雜度: 最糟跟 BFS 一樣  $O(b^d)$

- ◆ 雙向 A\*

- 時間複雜度  $O(b^{\lceil d/2 \rceil})$  (因有 close list)
- 空間複雜度  $O(b^{\lceil d/2 \rceil})$  (因有 close list)

- Reference

- Deadlock introduction:

- ◆ <http://sokobano.de/wiki/index.php?title=Deadlocks>

- Deadlock algorithm:

- ◆ [http://sokobano.de/wiki/index.php?title=How\\_to\\_detect\\_dea](http://sokobano.de/wiki/index.php?title=How_to_detect_dea)

[dlocks](#)

- 遊戲複雜度部分與 R06922019 徐誌鴻, R06922025 林家緯, R06922034 王皓正，共同討論的結果