Backtracking

การอบรมคอมพิวเตอรโอลิมปิก สอวน. ค่ายที่ 2
ปีการศึกษา 2564

Pisit Makpaisit

COMPUTER SCIENCE

Assignment 0

เขียนโปรแกรมแบบ recursion เพื่อหาค่าที่มากที่สุดในอาร์เรย์

Assignment 1

เขียนโปรแกรมแบบ recursion เพื่อหาค่าของลำดับ fibonacci ลำดับที่ n

Assignment 2

เขียนโปรแกรมแบบ recursion เพื่อแสดง permutation ทั้งหมดของสตริง “123456” เช่น “156423”, “512436”, “624513” เป็นต้น

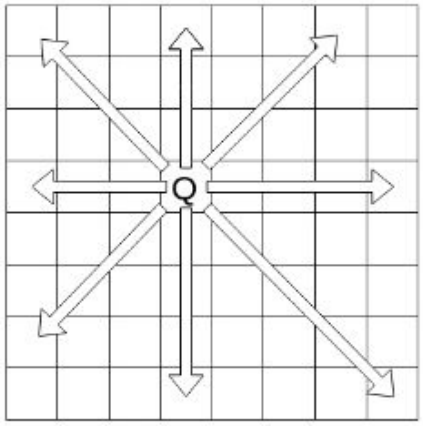
Backtracking

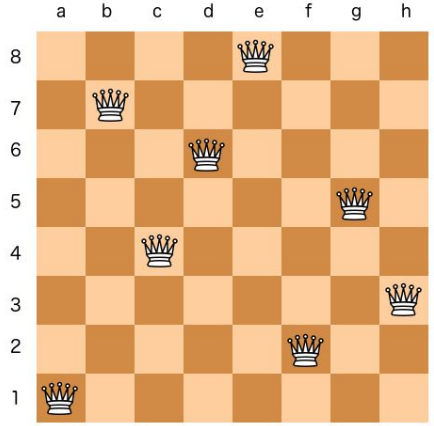
- เทคนิคในการแก้ปัญหาโดยสร้างคำตอบที่น่าจะเป็นไปได้ทีละตัว
- หากไม่ใช่คำตอบก็จะกำจัดทิ้ง และสร้างจนกว่าจะเจอคำตอบที่ตรงเงื่อนไข
- ทำงานคล้ายกับ brute force
- มักใช้การเขียนแบบ recursive
- ในทางทฤษฎีความซับซ้อนของเวลามักเป็นเอกซ์โพเนนเชียลหรือแฟกทอเรียล
- แต่สามารถใช้ได้ในทางปฏิบัติเนื่องจากเราสามารถตัดบางเส้นทางออกไป เพื่อช่วยลดรูปแบบที่ต้องพิจารณาลงได้

N-Queens Problem

ปัญหา N-Queens

ให้กระดานหมากรุกขนาด n x n ช่อง ต้องการวางควีนทั้งหมด n ตัวในกระดาน โดยที่ไม่มีควีนตัวใดตัวหนึ่งกินควีนอีกตัวหนึ่งได้ สามารถทำได้หรือไม่



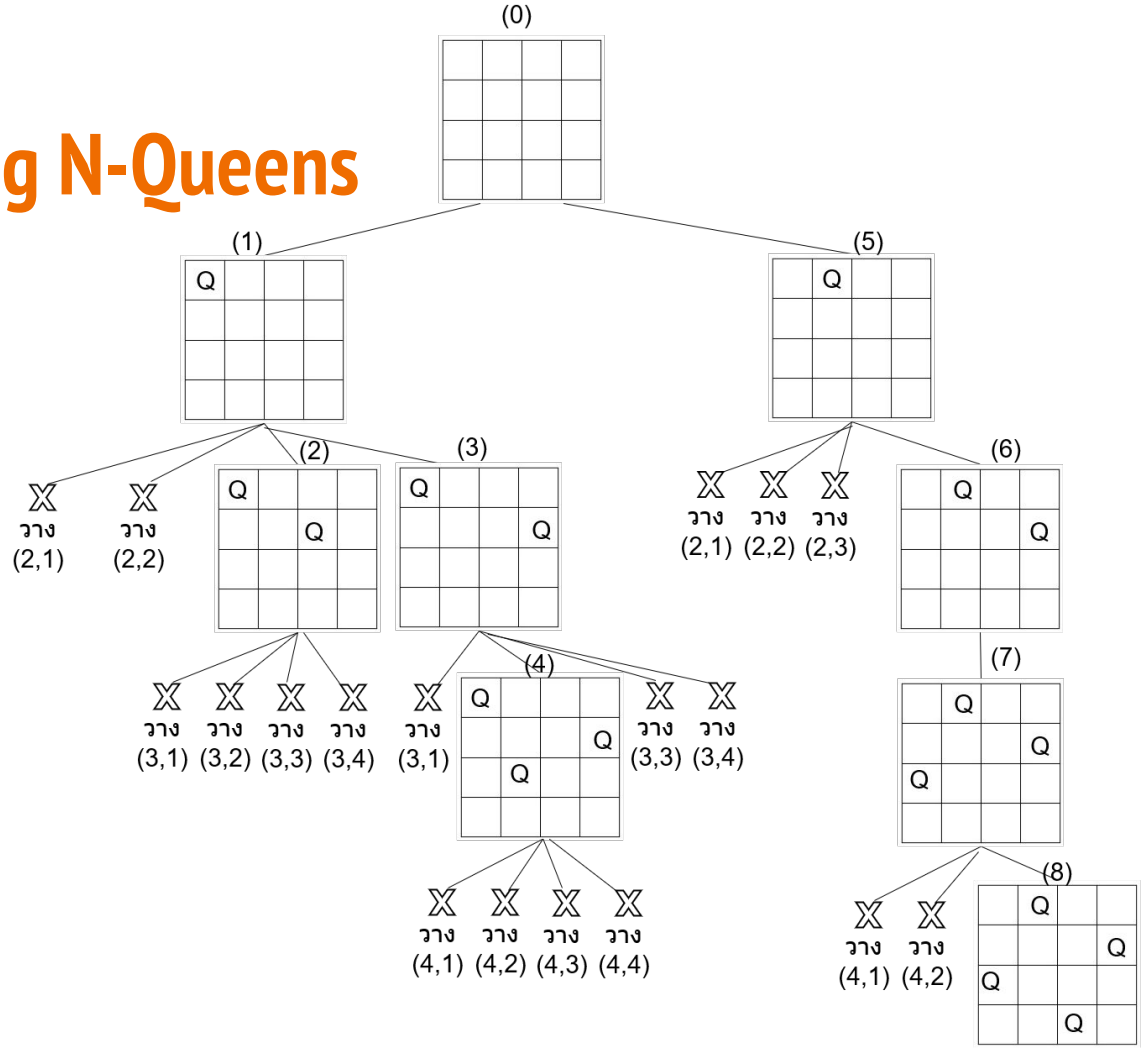


Observations

- ไม่สามารถวางควีนแต่ละตัวในแถวเดียวกันได้
  - ควีนแต่ละตัวจะต้องอยู่คนละแถวอย่างแน่นอน
- ไม่สามารถวางให้อยู่ในหลักเดียวกัน
- ไม่สามารถวางแนวทแยงเดียวกันได้
- ไม่ควรวางที่มุม ?

Recursive Backtracking N-Queens

สมมติให้ n = 4



Recursive Backtracking N-Queens

- เหมือนจะต้องค้นหาเป็นจำนวนมากจึงจะได้คำตอบ (n!)
- แต่สามารถตัดสถานะที่เป็นไปไม่ได้ออก ลดการพิจารณาได้เป็นจำนวนมาก
- เรียกว่าการ prune หรือ pruning
- หากเราสามารถ prune ได้ตั้งแต่ในชั้นแรกๆ ก็จะยิ่งทำให้สถานะที่ต้องพิจารณาลดลงไปมากยิ่งขึ้น

Recursive Backtracking N-Queens

Algorithm 8 recursiveNQueens

```
1: function nQueens(Q, n)
2:     if |Q| = n then                ▷ เมื่อขนาดของ Q เท่ากับ n แปลว่าได้คำตอบแล้ว และเก็บอยู่ใน Q
3:         Print solution from Q
4:     else
5:         for i = 0 to n do
6:             if can place another queen at i in the next row then    ▷ ถ้าวางช่องที่ i ในแถวถัดไปได้
7:                 Add i to Q
8:                 nQueens(Q, n)
9:                 Remove i from Q
```

Subset Sum Problem

ปัญหา Subset Sum

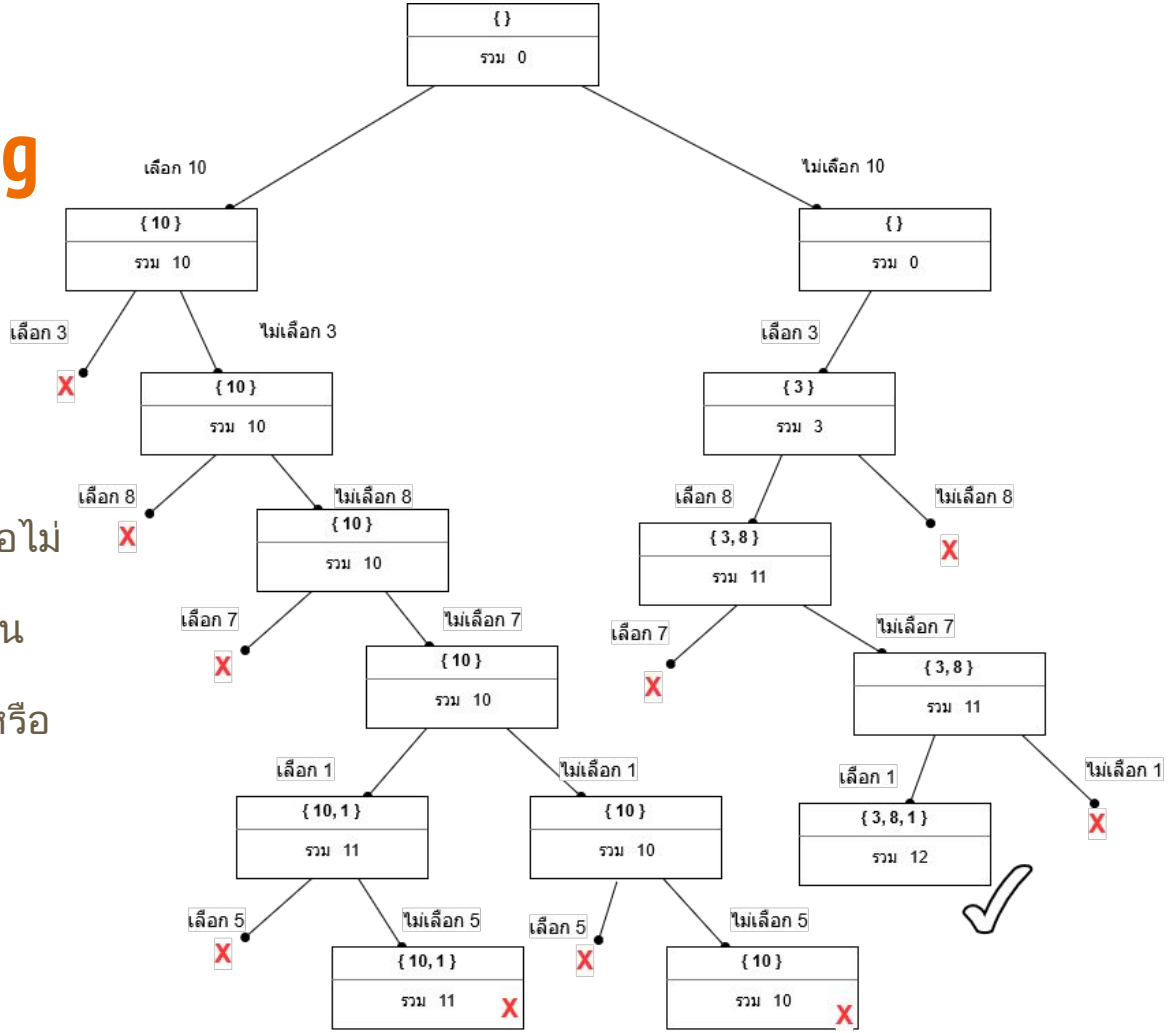
ให้เซต S = {a_1, a_2, a_3, ..., a_n} เป็นเซตของจำนวนเต็ม n ตัว มี subset ใดของ S หรือไม่ ที่มีผลรวมของสมาชิกเท่ากับจำนวนเต็ม k

Brute Force Subset Sum

- สร้าง subset ทุกรูปแบบ (มีทั้งหมด 2^n เซ็ต)
- ดูว่ามี subset ไหนที่ผลรวมเท่ากับ k หรือไม่
- $O(2^n)$

Recursive Backtracking Subset Sum

- เริ่มจากสับเซตที่ไม่มีสมาชิกเลย
- จะนำสมาชิกของเซ็ตตัวแรกมาใส่ใน subset หรือไม่ (มี 2 ทางคือเลือกหรือไม่เลือก)
- จะนำสมาชิกของเซ็ตตัวถัดไปมาใส่ใน subset หรือไม่
- แต่ละขั้นตอนให้เช็คว่าผลรวมได้ k หรือไม่
- ถ้าเกิน k หรือใช้สมาชิกจนครบ ให้ backtrack กลับไป



Recursive Backtracking Subset Sum

Algorithm 9 backtrackingSubsetSum

```
1: function subsetSum(S, k, S', i)
2:     if ∑S' = k then return True
3:     else if |S| = i then return False
4:     else
5:         if subsetSum(S, k, S', i + 1) then return True
6:         if subsetSum(S, k, S' ⋃{a_i}, i + 1) then return True
7:         return False
```

S' เป็นเซตสำหรับเก็บคำตอบของตัวเลขที่รวมเข้าไปเพื่อพิจารณา และ i เป็นตำแหน่งของสมาชิกใน S ที่กำลังพิจารณา

Assignment 3

เขียนโปรแกรมสำหรับแก้ปัญหา n-queens ด้วยวิธีการ recursive backtracking โดยให้ใส่ค่า n เข้าไปในโปรแกรม และแสดงรูปแบบการวาง queen ที่ถูกต้องมา 1 รูปแบบ ถ้าไม่มีให้แสดงข้อความว่า “No solution”

Assignment 4

เขียนโปรแกรมสำหรับแก้ปัญหา subset sum ด้วยวิธีการ recursive backtracking โดยโปรแกรมสามารถใส่ค่า n ที่เป็นจำนวนของสมาชิกในเซ็ต และตามด้วยค่าของสมาชิกทั้งหมด n ตัว จากนั้นใส่ k และแสดง subset ทั้งหมด ที่มีผลรวมเท่ากับ k (ถ้าไม่มีไม่ต้องพิมพ์อะไรออกมาทางหน้าจอ)

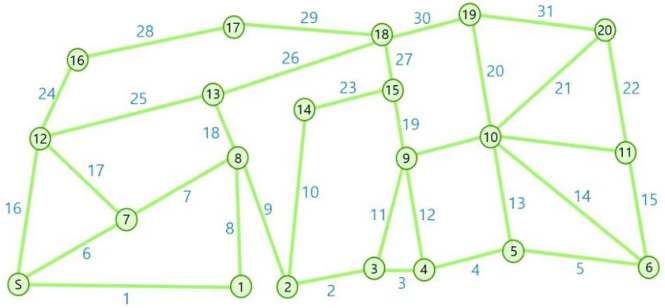
Branch and Bound

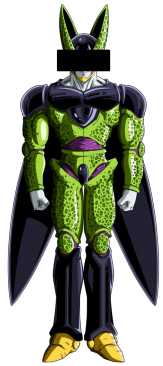
- Optimization problem คือปัญหาที่ต้องการหาคำตอบที่ดีที่สุด
- Optimization problem สามารใช้ backtracking ในการหาคำตอบได้เช่นกัน
- Branch and bound เป็นหนึ่งในเทคนิคการ prune สำหรับ optimization problem
- ใช้การจำ bound ของค่าที่ดีที่สุด และดูว่าสถานะใดที่ยังสามารถทำให้ดีกว่านี้ได้บ้าง

Traveling Salesman Problem

ปัญหา TSP

มีเมืองทั้งหมด n เมือง และมีเซลคนหนึ่งที่ต้องการเดินทางไปขายของในทุกเมือง และจะไปแต่ละเมืองเพียงครั้งเดียวเท่านั้น โดยเริ่มต้นเดินทางจากเมืองที่กำหนดไว้และเมื่อเดินทางเสร็จก็ให้กลับมายังเมืองเดิม ค่าเดินทางจากเมือง i ไปเมือง j เป็น c_{ij} หาค่าเดินทางที่น้อยที่สุด



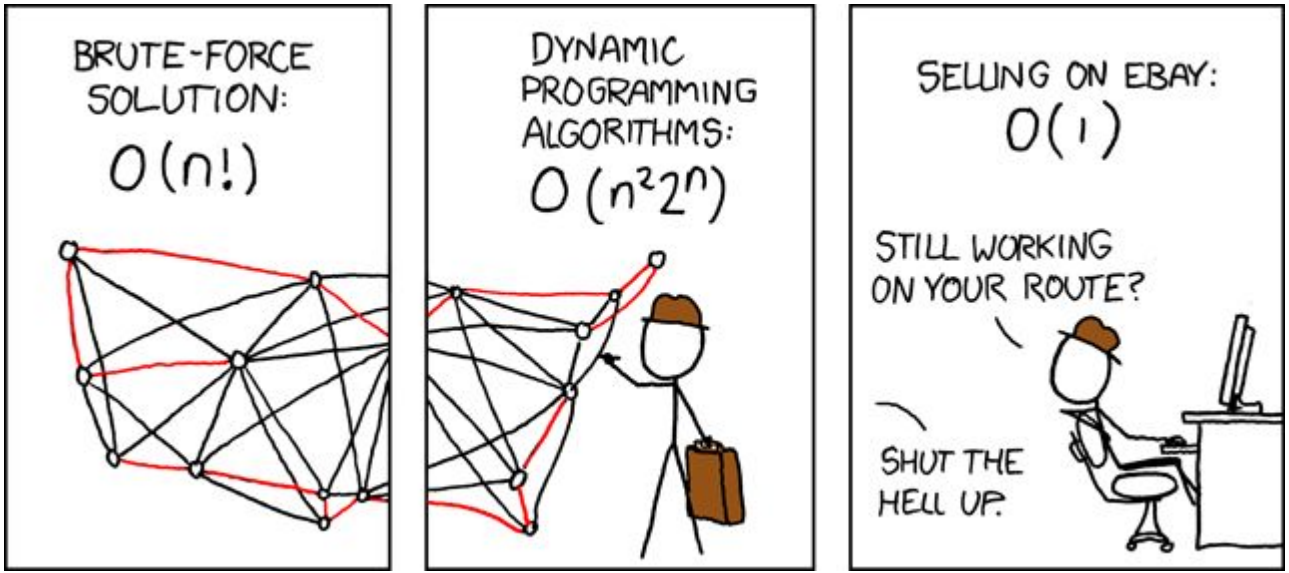


https://www.researchgate.net/publication/338071918_CityGML-Based_Road_Information_Model_for_Route_Optimization_of_Snow-Removal_Vehicle

Traveling Salesman Problem Solutions

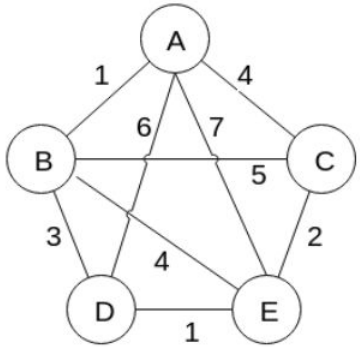
- Brute force
  - สร้าง permutation ของเมืองทั้งหมด
  - หาว่าเส้นทางใดเป็นไปได้
  - หาเส้นทางที่เป็นไปได้และใช้ cost น้อยสุด
  - Θ(n!)
- Greedy
  - จากเมืองเริ่มต้นเลือกเส้นทางที่ cost น้อยสุด
  - เลือก cost น้อยสุดไปเรื่อยๆ
  - เส้นทางที่ cost น้อยตอนแรกอาจจะนำไปสู่ cost มากในตอนหลัง (ไม่ optimal)

Traveling Salesman Problem Solutions



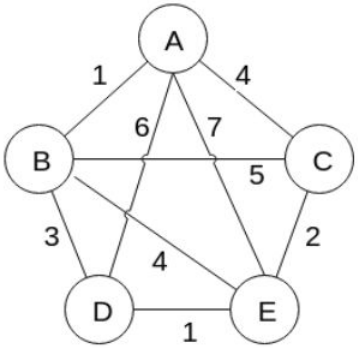
https://www.explainxkcd.com/wiki/index.php/399:_Travelling_Salesman_Problem

Branch and Bound



	A	B	C	D	E
A		①	④	6	7
B	①		5	③	4
C	④	5			②
D	6	③			①
E	7	4	②	①	

Branch and Bound



Tour: เรียกเส้นทางการเดินทางจากเมืองเริ่มต้นไปยังทุกเมือง และกลับมาเมืองเดิมว่า Tour เช่น

- A->E->D->B->C->A
- A->B->C->E->D->A
- A->C->B->D->E->A เป็นต้น

ไม่ว่า Tour จะเรียงลำดับการเดินทางอย่างไร แต่ละ Tour จะมีระยะทางรวมเท่ากับระยะของ in edge และ out edge ของทุกเมืองรวมกันแล้วหารด้วย 2 เสมอ

Branch and Bound

	A	B	C	D	E
A		①	④	6	7
B	①		5	③	4
C	④	5			②
D	6	③			①
E	7	4	②	①	

Lower bound ของ Tour :
ไม่ว่า tour ใดๆ จะต้องมีระยะทางรวมไม่น้อยไปกว่านี้

Lower bound เริ่มต้นคำนวณได้จาก
ผลรวมของ 2 edge ที่มีค่าน้อยที่สุดของแต่ละ vertex

$\lceil [(1 + 4) + (1 + 3) + (4 + 2) + (3 + 1) + (2 + 1)] / 2 \rceil = 11$

* หมายความว่าไม่ว่า Tour ใดๆ ก็จะไม่สามารถมี cost รวมน้อยกว่า 11 ได้**เด็ดขาด**

Branch and Bound

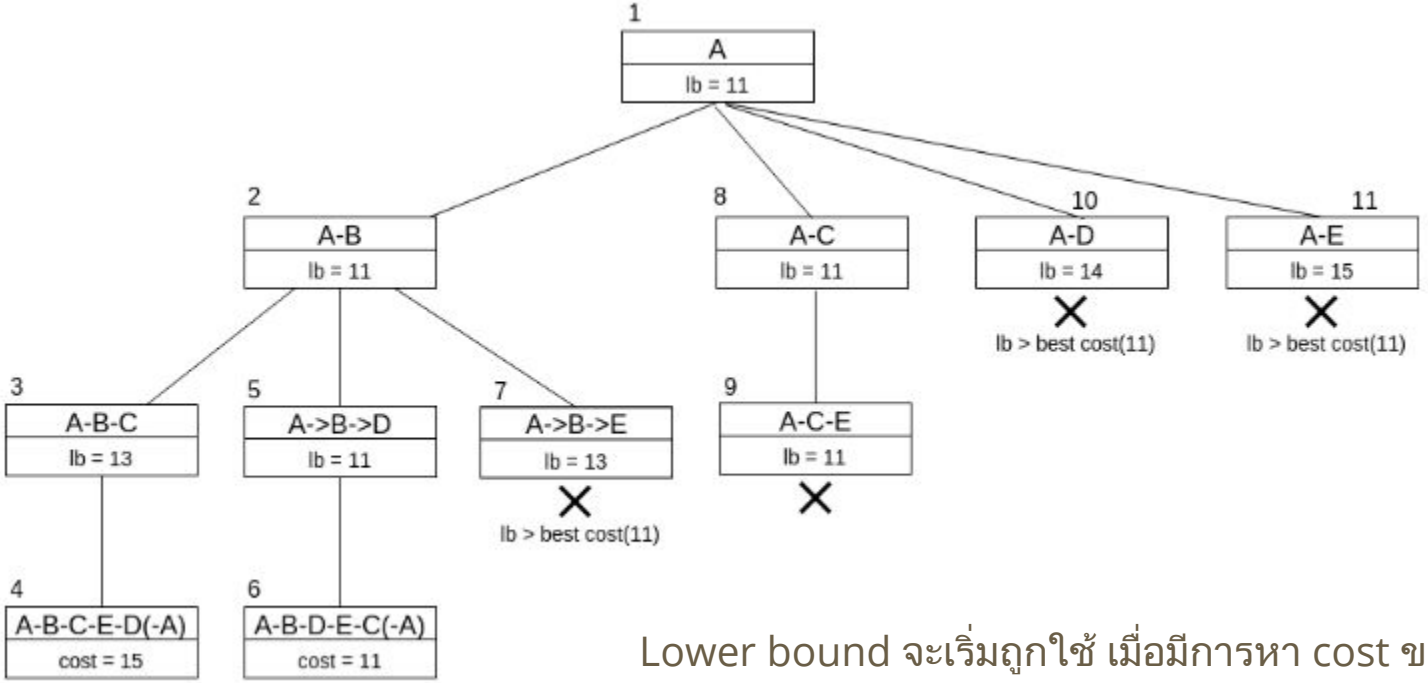
	A	B	C	D	E
A		①	④	6	7
B	①		5	③	4
C	④	5			②
D	6	③			①
E	7	4	②	①	

สำหรับเส้นทางที่เลือกเมือง A ไปยัง D ไปแล้ว Lower bound จะเป็น

⌈[(1 + 6) + (1 + 3) + (4 + 2) + (6 + 1) + (2 + 1)] / 2⌉= 14

ตัวเลข lower bound นี้คือค่าเดินทางต่ำสุดที่จะเป็นไปได้ของเส้นทาง (หรือส่วนของเส้นทาง) ถ้าหากเราค้นไปเจอเส้นทางที่สั้นกว่า lower bound ของสถานะเหล่านี้เมื่อไหร่ก็หมายความว่าสามารถตัดสถานะนั้นออกไปจากการค้นหาได้ทันที เพราะไม่มีทางที่ไปต่อแล้วจะได้เส้นทางน้อยกว่านี้

Branch and Bound



Lower bound จะเริ่มถูกใช้ เมื่อมีการหา cost ของบาง solution ได้แล้ว

Assignment 5 - 0/1 Knapsack Problem

จงอธิบายว่าสำหรับปัญหา 0/1 Knapsack problem สามารถใช้วิธีการ branch and bound ได้อย่างไร เพื่อลดเวลาในการหาคำตอบที่ดีที่สุด

8-Puzzle Problem

ปัญหา 8-Puzzle

เกมที่มีตารางขนาด 3 x 3 และมีแผ่นตัวเลข 1-8 วางอยู่ สามารเลื่อนแผ่นตัวเลขได้ 4 ทิศทาง ขึ้น ลง ซ้าย ขวา จากการวางในตอนแรก (Initial State) จะต้องเลื่อนแผ่นตัวเลขอย่างไรให้กลายเป็นแบบที่ต้องการ (Goal State)

	3	7
5	1	6
2	8	4

Initial State

1	2	3
4	5	6
7	8	

Goal State

State Space Search

การค้นหาใน State Space (หา Solution ที่ตรงกับเงื่อนไขที่ต้องการ)

State - สถานะที่เป็นไปได้

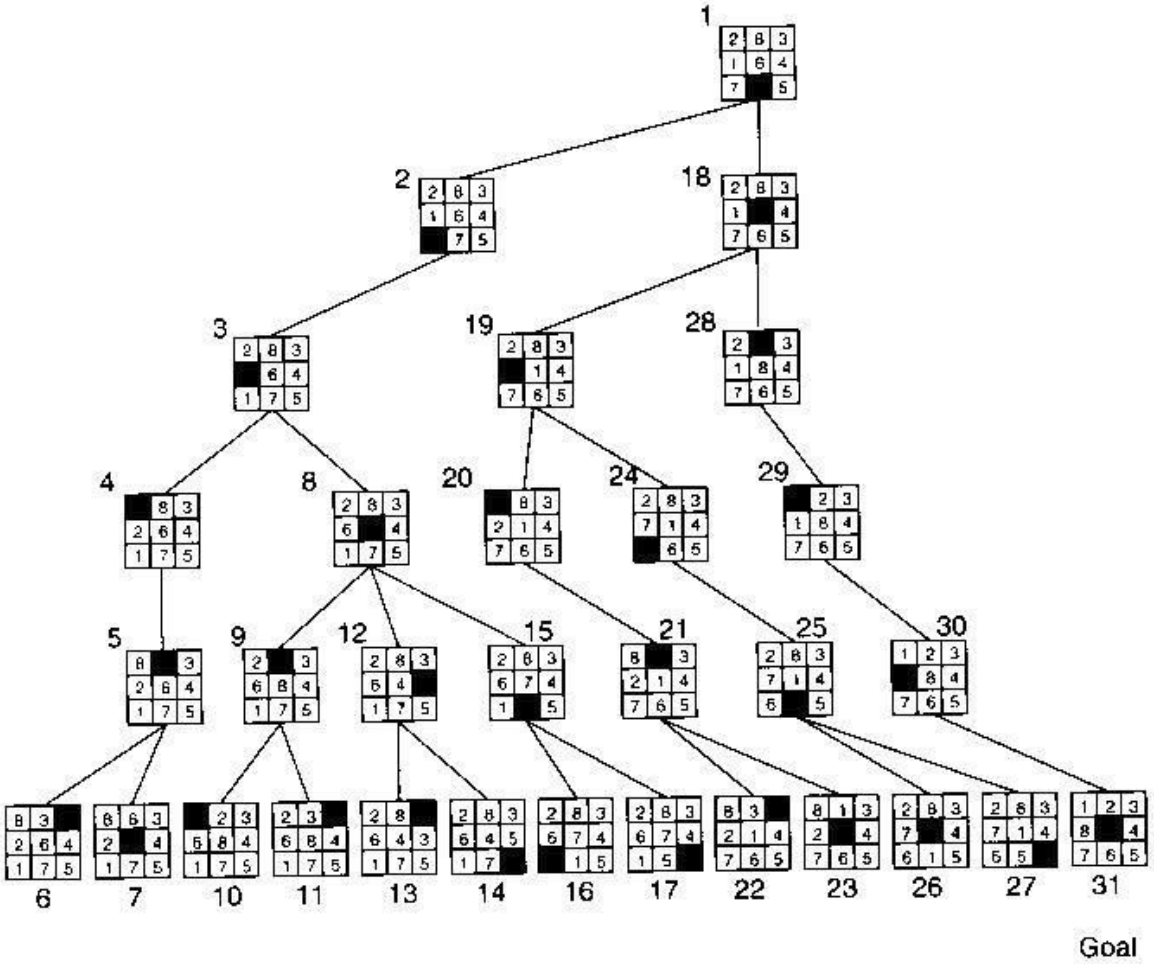
Operator / Move - action ที่เป็นไปได้ในการเปลี่ยนจากสถานะหนึ่งไปยังอีกสถานะหนึ่ง

State space - สถานะที่เป็นไปได้ทั้งหมด

Start state / Initial state - สถานะเริ่มต้น (มีได้รูปแบบเดียว)

Goal state - สถานะเป้าหมายที่ต้องการ (อาจมีได้หลายรูปแบบ)

State Space Search



https://i.cs.hku.hk/~kpchan/cs23270/3.problem-solving/problem-solving.html

State Space Search

- ถ้ามอง state space ทั้งหมดเป็น graph / tree
- ให้ initial state เป็นจุดเริ่มต้นในการค้นหา
- Backtracking เป็นเหมือน DFS (Depth First Search)
- สามารถใช้ BFS (Breadth First Search) ในการหาได้เช่นกัน
  - หาคำตอบที่ใกล้กับ initial state มากที่สุด
  - ใช้ loop + queue แทนการเขียนด้วย recursion

BFS State Space Search Algorithm

Algorithm 10 BreadthFirstSearch

```
1:  Open ← {StartState}
2:  Visited ← {}
3:  while Open not empty do
4:      q ← remove first state from Open
5:      if q is goal state then return Path from start to q or solution
6:      else
7:          Expand possible states S from q
8:          for each state n in S do
9:              if n ∉ Visited and n ∉ Open then
10:                 Open ← Open ⋃ {n}
11:         Visited ← Visited ⋃ {q}
12: return Not found solution
```

BFS State Space Search Algorithm

- Visited เป็น set ของ state ที่ไปมาแล้ว
  - ไม่ใช้ได้ถ้ามั่นใจว่าจะไม่มีการกลับไป state เดิมที่เคยไปมาแล้วแน่นอน
  - เช่น Tic-Tac-Toe
- Open เป็น set ของ state ที่กำลังจะพิจารณา และจะนำมา expand
- บรรทัดที่ 4 จะนำ state แรกที่อยู่ใน Open ออกมา
  - ใช้ Queue ในการ implement

Heuristic Search

Uninformed Search / Blind Search

- การค้นหาที่ไม่ได้ใช้ข้อมูลหรือความรู้เพิ่มเติม อาศัยการดูไปทีละ state
- เช่น การค้นหาแบบปกติด้วย DFS และ BFS
- ใช้กับ state space ที่ไม่ใหญ่มาก สามารถเจอคำตอบได้เร็ว

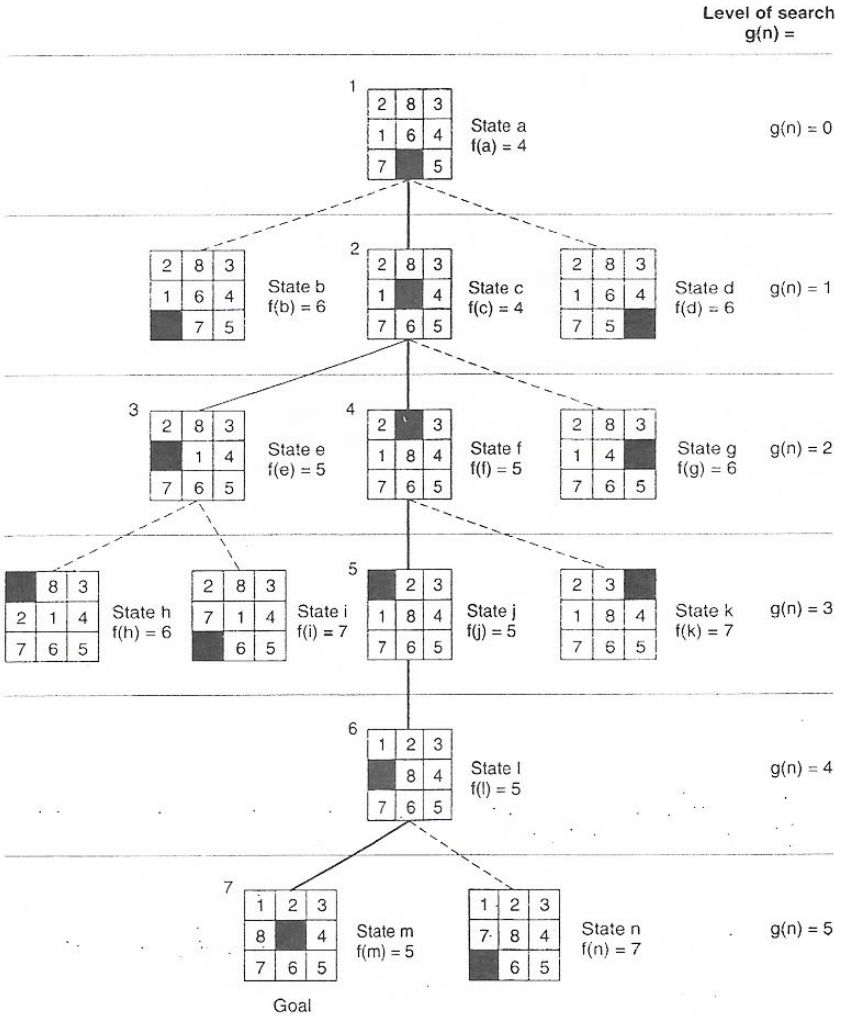
Informed Search / Heuristic Search

- ใช้ความรู้ความเข้าใจในตัวปัญหา (ของคนเขียนโปรแกรม) เพิ่มลงไปในโปรแกรม
- ค้นหาด้วยกฎที่เพิ่มเข้าไป

Best-First Search

- เลือก expand state จาก state ที่คิดว่าใกล้กับ goal state มากที่สุดก่อน
- ใช้ Evaluation function f(n) ในการคำนวณว่าใกล้ goal state มากแค่ไหน
- Evaluation function f(n) คือส่วนของความรู้แบบ heuristic ที่เราใส่เข้าไปในโปรแกรม

Best-First Search



Evaluation Functions Example (8-Puzzle Problem)

1	2	4
5	3	
6	7	8

State (A)

4	1	2
7	6	3
	5	8

State (B)

3	6	4
2	5	
7	1	8

State (C)

1	2	3
4	5	6
7	8	

Goal State

State ใดใกล้เคียงกับ Goal State มากที่สุด ? (A), (B) หรือ (C)

Evaluation Functions Example (8-Puzzle Problem)

- Number of Misplaced Tiles - จำนวนของ tile ที่ไม่อยู่ในตำแหน่งที่ต้องการ
- ผลรวมของ Manhattan distance
- ผลรวมของ Euclidean distance
- n-swap - ถ้าสามารถสลับตำแหน่งของ 2 tiles ใดๆ ได้ (ให้ช่องว่างเป็นเหมือน tile หนึ่ง) จำนวนที่น้อยที่สุดในการสลับเพื่อไปยัง goal state

Number of Misplaced Tiles

3	6	4
2	5	
7	1	8

Current State

1	2	3
4	5	6
7	8	

Goal State

Tile	Missing?
1	1
2	1
3	1
4	1
5	0
6	1
7	0
8	1

Number of missing tiles = 6

Number of Misplaced Tiles

1	5	3
2	6	4
7	8	

Current State

1	2	3
4	5	6
7	8	

Goal State

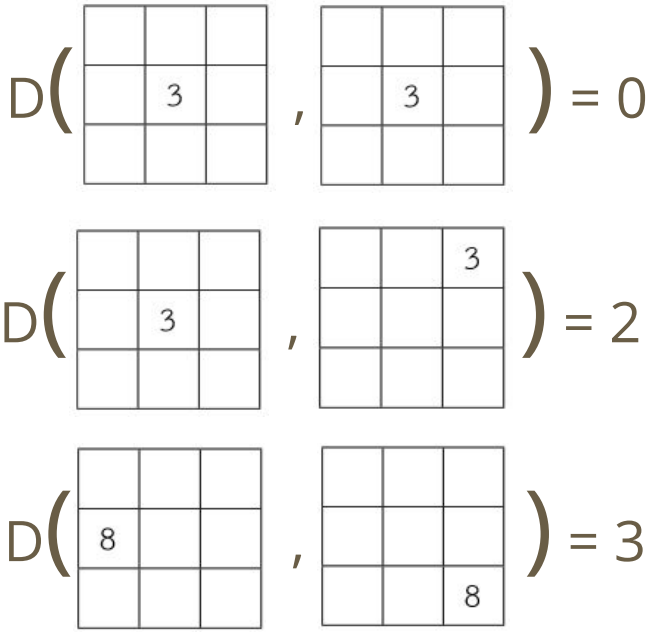
Tile	Missing?
1	
2	
3	
4	
5	
6	
7	
8	

Number of missing tiles = ?

Manhattan Distance

|x1 - x2| + |y1 - y2|

- x1 คอลัมภ์ของสถานะแรก
- x2 คอลัมภ์ของสถานะสอง
- y1 แถวของสถานะแรก
- y2 แถวของสถานะสอง



Sum of Manhattan Distance

3	6	4
2	5	
7	1	8

Current State

Tile	Distance
1	3
2	2
3	2
4	3
5	0
6	2
7	0
8	1

1	2	3
4	5	6
7	8	

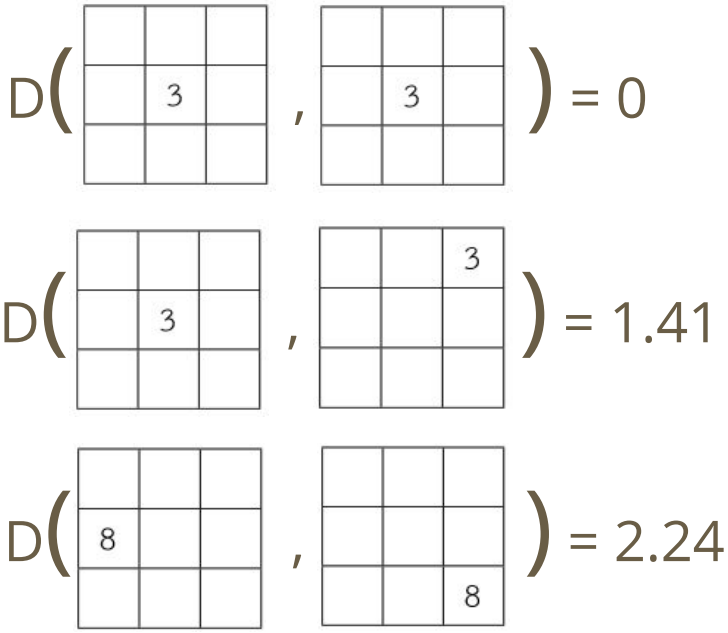
Goal State

Total = 3 + 2 + 2 + 3 + 0 + 2 + 0+ 1 = 13

Euclidean Distance

$$\sqrt{(x1-x2)^2+(y1-y2)^2}$$

- x1 คอลัมภ์ของสถานะแรก
- x2 คอลัมภ์ของสถานะสอง
- y1 แถวของสถานะแรก
- y2 แถวของสถานะสอง



Evaluation Function

f(n) = g(n) + h(n)

- n - state ใดๆ ใน state space
- g(n) - cost ที่ใช้จาก initial state ไปยัง state n
  - สำหรับ 8-Puzzle ใช้เป็น depth ของ state หรือระยะจาก initial state จนถึง state n
- h(n) - heuristic estimate cost จาก state n ไป goal state
  - เช่นตัวอย่างฟังก์ชันที่เสนอมา

ไม่มี g(n) ได้หรือไม่?

Best-First Search Algorithm

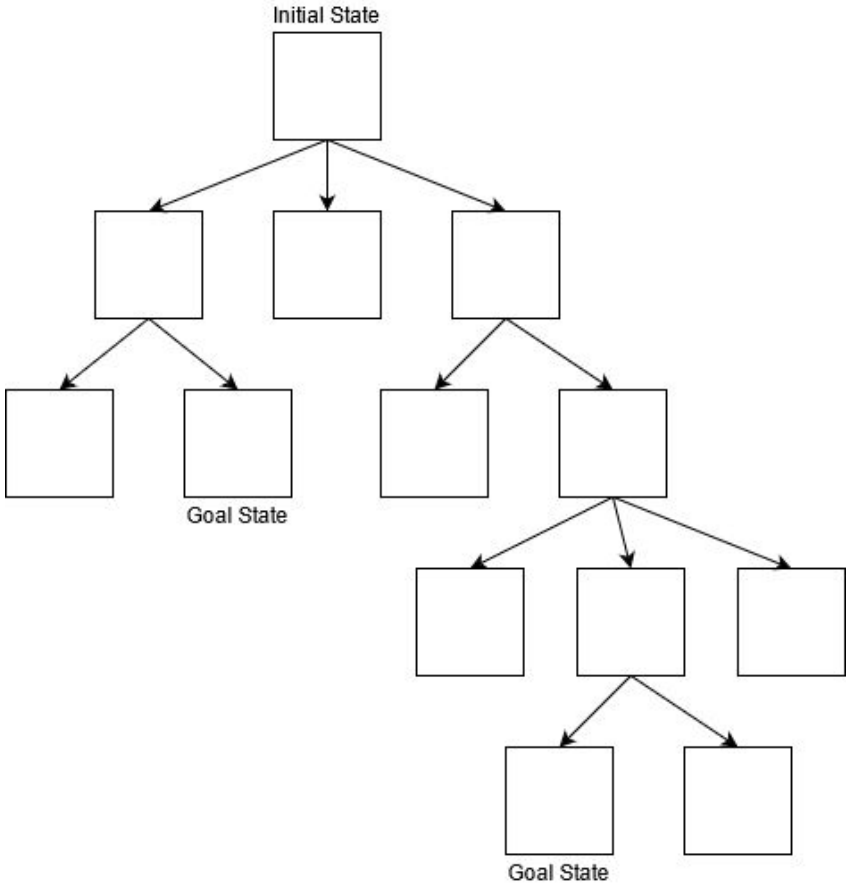
Algorithm 11 BestFirstSearch

```
1:  Open ← {StartState}
2:  Visited ← {}
3:  while Open not empty do
4:      q ← remove minimum f(n) state from Open
5:      if q is goal state then return Path from start to q or solution
6:      else
7:          Expand possible states S from q
8:          for each state n in S do
9:              if n ∉ Visited and n ∉ Open then
10:                 Open ← Open ⋃ {n}
11:         Visited ← Visited ⋃ {q}
12: return Not found solution
```

- ใช้ priority queue ในการเก็บ Open เพื่อให้ดึงสถาน n ที่ f(n) น้อยที่สุดออกมาได้เร็ว

Best-First Search Algorithm

- มีโอกาสที่จะเจอ solution ที่ไม่ optimal หรือไม่?



Admissibility

- Search algorithm จะเรียกว่า admissibility ถ้าสามารถการันตีได้ว่าหา minimal path ไปสู่ goal state ได้ (ในกรณีที่มี solution)
  - Breadth First Search เป็น admissibility
- Evaluation function ที่คิดขึ้นมาจะต้อง admissible เพื่อให้การันตีได้ว่าสามารถหา minimal path / shortest path จาก initial state ไปยัง goal state ได้

A* Algorithm

สมมติ Evaluation function f*(n) ขึ้นมา โดยมีนิยามดังนี้

f*(n) = g*(n) + h*(n)

- n - state ใดๆ ใน state space
- g*(n) - cost ของ shortest path จาก initial state ไปยัง state n
- h*(n) - actual cost ของ shortest path จาก state n ไปยัง goal state

f*(n) จะหมายถึง actual cost ของ optimal path จาก initial state ไปยัง goal state เมื่อผ่าน state n

f(n) = g(n) + h(n)

- n - state ใดๆ ใน state space
- g(n) - cost ที่ใช้จาก initial state ไปยัง state n
- h(n) - heuristic estimate cost จาก state n ไป goal state

f(n) จะหมายถึง estimate cost จาก initial state ไปยัง goal state เมื่อผ่าน state n

A* Algorithm

กำหนด Evaluation function f(n) = g(n) + h(n)

- ถ้า h(n) ≤ h*(n) จะเรียก search algorithm ที่ใช้ evaluation function f(n) ว่า A* algorithm
- A* algorithm มีคุณสมบัติ admissibility
- A* algorithm การันตีว่าสามารถหา minimal path จาก initial state ไป goal state (ถ้ามี)

A* Algorithm

- **Breadth First Search** - h(n) = 0 เสมอ ดังนั้น h(n) ≤ h*(n)
- **Number of Misplaced Tiles** - จำนวนของแผ่นตัวเลขที่อยู่ผิดตำแหน่ง จะน้อยกว่าจำนวนครั้งที่น้อยที่สุดในการเลื่อนเพื่อให้เป็น goal state ดังนั้น h(n) ≤ h*(n)
- **Sum of Manhattan Distance** - การเลื่อน 1 ครั้งสามารถทำให้มี 1 tile เท่านั้นที่ใกล้ goal state มากขึ้น ดังนั้น h(n) ≤ h*(n)

Informedness

จะรู้ได้อย่างไรว่า heuristic ตัวไหนดีกว่า?

- Informedness เป็นคุณสมบัติที่ใช้บอกว่า heuristic ตัวไหนที่ดีกว่ากัน
- ดีกว่า = ให้ข้อมูลเยอะกว่า (inform กว่า) ซึ่งจะทำให้หา solution ได้เร็วขึ้น เพราะจะ expand state น้อยกว่า

สมมติให้มี heuristic h_1 และ h_2
ถ้า $h_1(n) \leq h_2(n)$ สำหรับทุก state n ใน state space จะบอกได้ว่า h_2 inform กว่า h_1

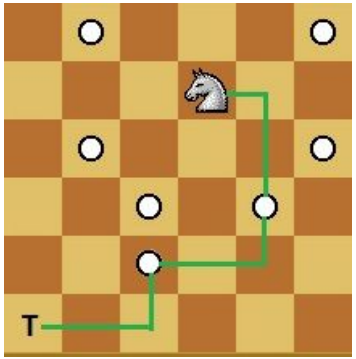
Informedness

Breadth First Search เป็น Admissible
Number of Misplaced Tiles เป็น Admissible

ดังนั้นทั้งคู่การันตีกว่า จะได้ minimal path เป็นคำตอบ แต่

- h(n) ของ Breadth First Search (ให้เป็น $h_1(n)$) เท่ากับ 0 เสมอ
- h(n) ของ Number of Misplaced Tiles ให้เป็น $h_2(n)$
- เนื่องจาก $0 \leq h_2(n)$
- ดังนั้น $h_1(n) \leq h_2(n)$
- บอกได้ว่า Number of Misplaced Tiles นั้น inform มากกว่า และหาคำตอบได้เร็วกว่า BFS

Assignment 6



Knight on Chessboard

กระดานหมากรุกขนาด n x n เขียนโปรแกรมที่รับตำแหน่งที่วางตัวม้า และตำแหน่งเป้าหมายที่ต้องการไป แสดงจำนวนการเดินที่น้อยที่สุดของม้าเพื่อไปตำแหน่งดังกล่าว

หากสามารถแสดงวิธีการเดินมาได้ ให้แสดงมาด้วย

References

1. Ahmed Shamsul Arefin.Art of Programming Contest. 2006.
2. Anany Levitin.Introduction to The Design & Analysis of Algorithms. Pearson Education, 2nd edition, 2007.
3. Steven S. Skiena.The Algorithm Design Manual. Springer, 2nd edition, 2008.
4. Jon Kleinberg & Éva Tardos.Algorithm Design. Pearson Education, 2006.
5. สมชาย ประสิทธิ์จูตระกูล.การออกแบบและวิเคราะหอัลกอริทึม. ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย, พิมพ์ครั้งที่ 4, 2553