เทคนิคการค้นหาคำตอบ

ทัศนวรรณ ศูนย์กลาง

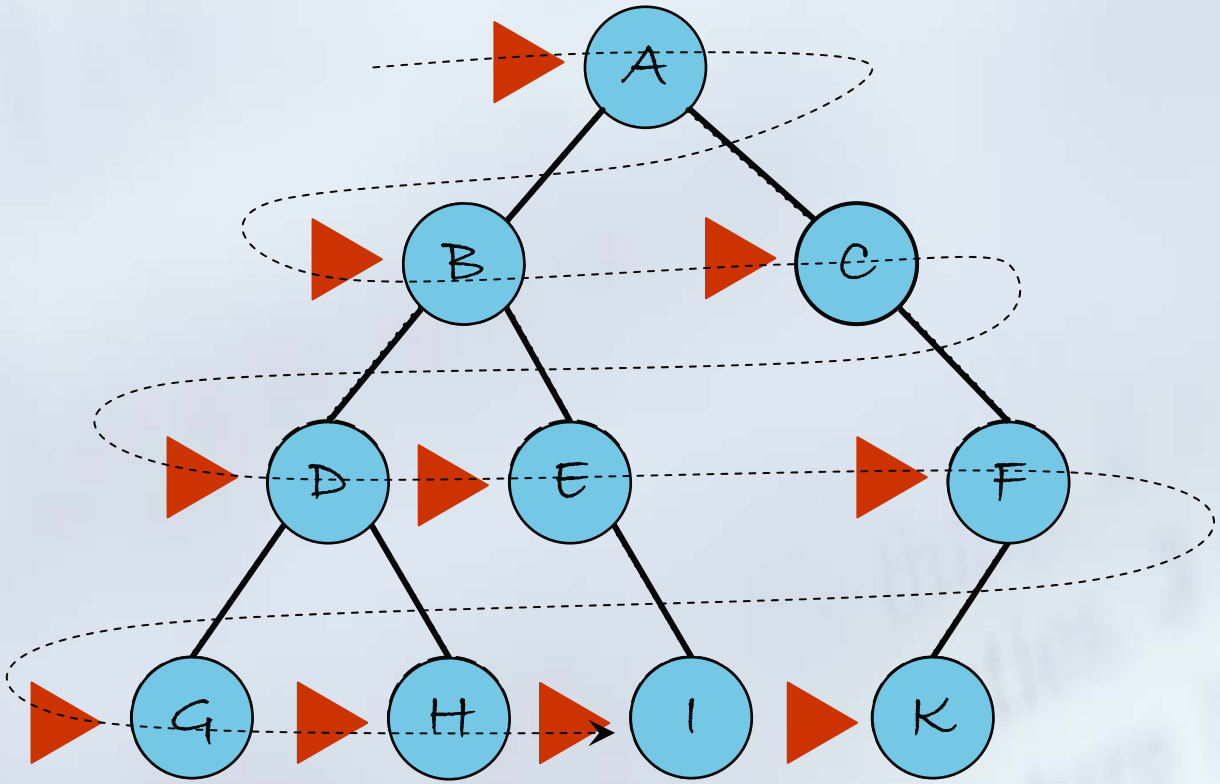
เนื้อหาการบรรยาย

- เทคนิคการค้นหา
  - การค้นหาแนวกว้างก่อน
  - การค้นหาแนวลึกก่อน
- การค้นหาแบบฮิวริสติก
  - การค้นหาแบบปีนเขา
  - การค้นหาแบบจำลองการอบเหนียว
  - การค้นหาแบบดีที่สุดก่อน
  - การค้นหาแบบเอ-สตาร์

เทคนิคการค้นหา : Search techniques

- การค้นหาแบบบอด (Blind search)
  - การค้นหาแบบทั้งหมด (Exhaustive search)
  - การค้นหาแบบบางส่วน (Partial search)
    - **การค้นหาแนวกว้างก่อน (Bread-first search)**
    - **การค้นหาแนวลึกก่อน (Depth-first search)**
- การค้นหาแบบฮิวริสติก (Heuristic search)
  - **การค้นหาแบบปีนเขา (Hill climbing)**
  - **การค้นหาแบบอบเหนียวจำลอง (Simulated annealing)**
  - **การค้นหาแบบดีที่สุดก่อน (Best-first search)**
  - **การค้นหาแบบเอ-สตาร์ (A* algorithm)**
  - การค้นหาแบบอื่นๆ

การค้นหาแนวกว้างก่อน : Breadth-first search



BFS Algorithm

1. *NODE-LIST* := {initial state}.

2. UNTIL a goal state is found or *NODE-LIST* is empty DO:

   2.1 Remove the first element from *NODE-LIST* and call it *E*.

   IF *NODE-LIST* is empty THEN quit.

   2.2 FOR each operator matching *E* DO:

   2.2.1 Apply the operator to generate a new state.

   2.2.2 IF the new state is a goal state THEN quit and return this state.

   ELSE add the new state to the end of *NODE-LIST*.

ข้อดีและข้อเสียของ BFS

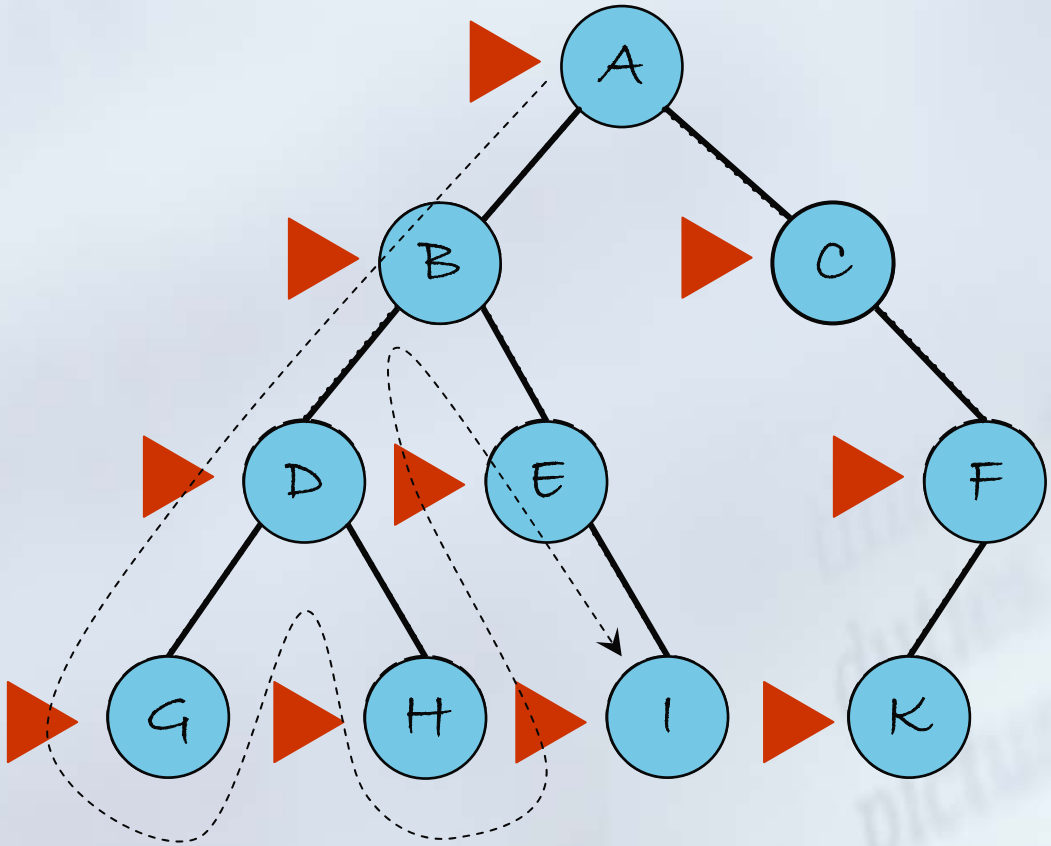
ข้อดี

- ถ้ามีคำตอบ ประกันได้ว่าจะพบคำตอบแน่
- คำตอบที่พบจะเป็นเส้นทางที่สั้นที่สุด
- ไม่ติดในเส้นทางที่ลึกมากๆ โดยไม่พบคำตอบ

ข้อเสีย

- ใช้หน่วยความจำมาก เพราะจำนวนโหนดในแต่ละระดับจะเพิ่มแบบ exponential
- เสียเวลามากในกรณีที่คำตอบอยู่ในระดับลึก
- ถ้าต้นไม้มีสถานะซ้ำๆ กันมาก จะทำให้ต้องสำรวจมากตามไปด้วย

การค้นหาแนวลึกก่อน : Depth-first search



DFS Algorithm

1. IF initial state = goal state THEN quit and return success.

   ELSE DO the following UNTIL success or failure is signaled:

   1.1 Generate a successor, *E*, of the initial state.

   IF there are no more successors THEN signal failure.

   1.2 Call Depth-First Search with *E* as the initial state.

   1.3 IF success is returned THEN signal success.

   ELSE continue in this loop.

ข้อดีและข้อเสียของ DFS

ข้อดี

- ใช้หน่วยความจำน้อยกว่า BFS เพราะไม่ต้องกระจายโหนดมาก เก็บเฉพาะโหนดในเส้นทางปัจจุบัน
- ถ้าคำตอบอยู่ในระดับลึก จะพบคำตอบโดยไม่ต้องค้นหามากเกินจำเป็น

ข้อเสีย

- ในบางภาษาโปรแกรม เช่น prolog อาจจะทำให้เกิด overflow ได้ ควรจำกัดความลึก
- อาจติดในเส้นทางที่ลึกมากๆ ทำให้เสียเวลา กรณีที่คำตอบอยู่ในระดับบนๆ

State space search : Block world

- ตัวกระทำการ คือ

1. กล่อง X ไม่มีกล่องอื่นทับ -> วาง X บนโต๊ะ

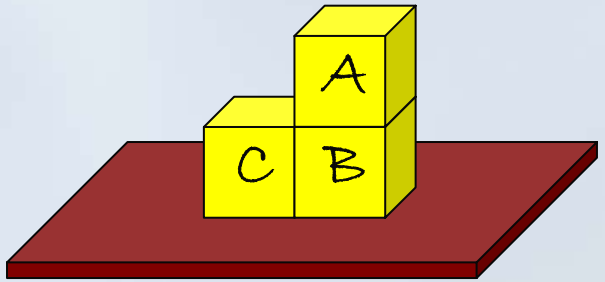
   CLEAR(X) -> ON(X,TABLE)

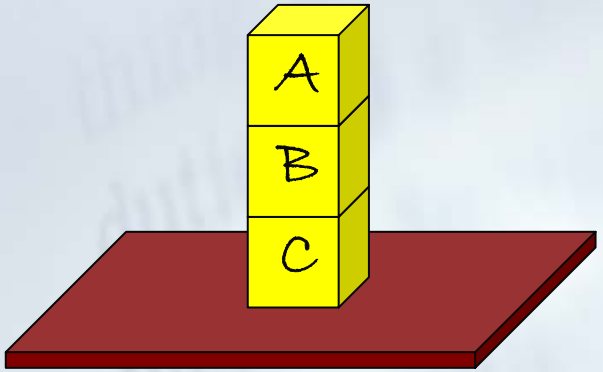
2. กล่อง X และ Y ไม่มีกล่องอื่นทับ -> วาง X บน Y

   CLEAR(X) & CLEAR(Y) -> ON(X,Y)

- X และ Y เป็นตัวแปร แทนที่ได้ด้วย A, B, C
- ไม่สนใจลำดับของกล่องในแนวเดียวกัน

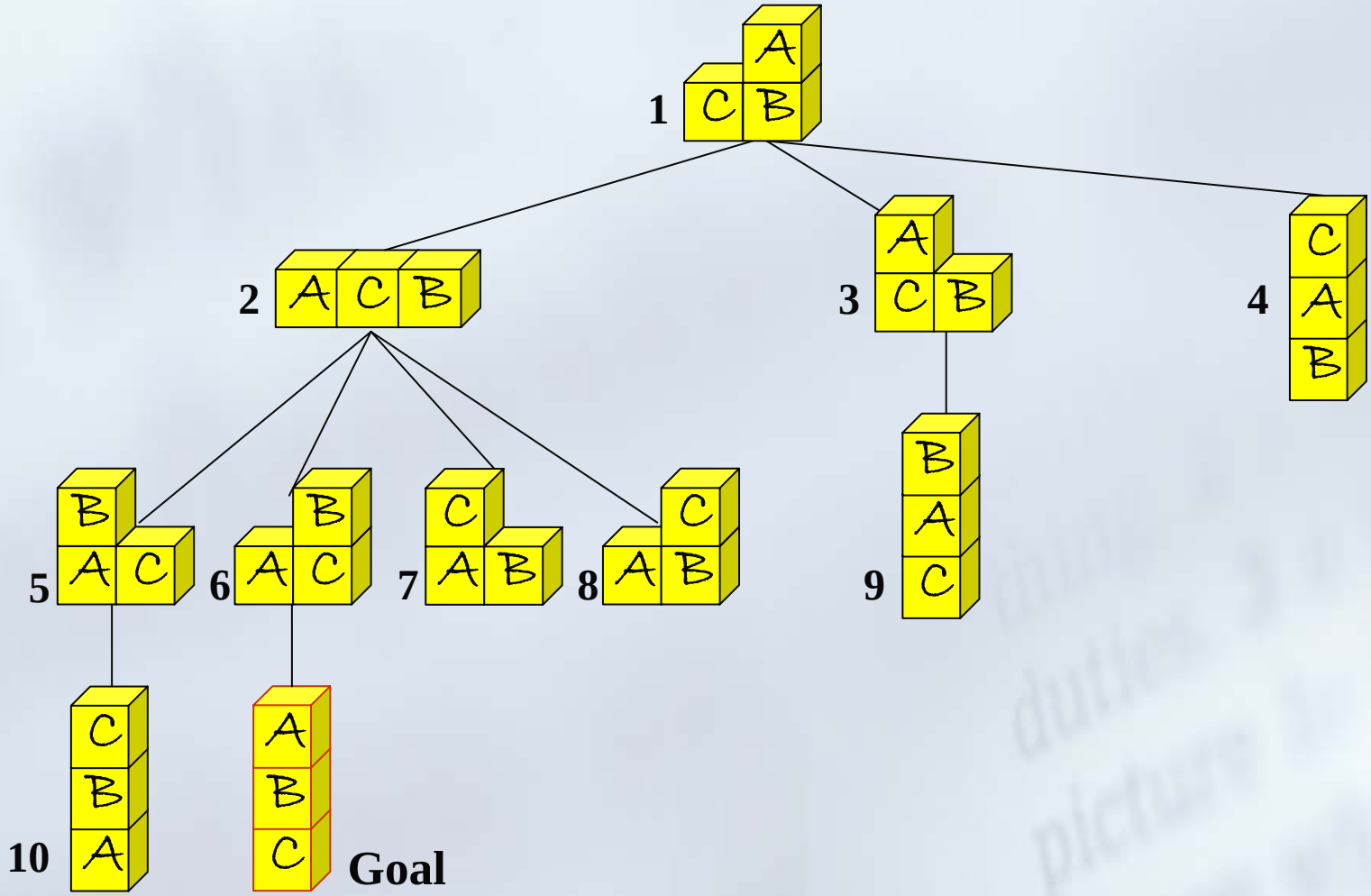
  เช่น สถานะ 'A B C' เท่ากับสถานะ 'C B A'
- ไม่สร้างสถานะซ้ำเดิม



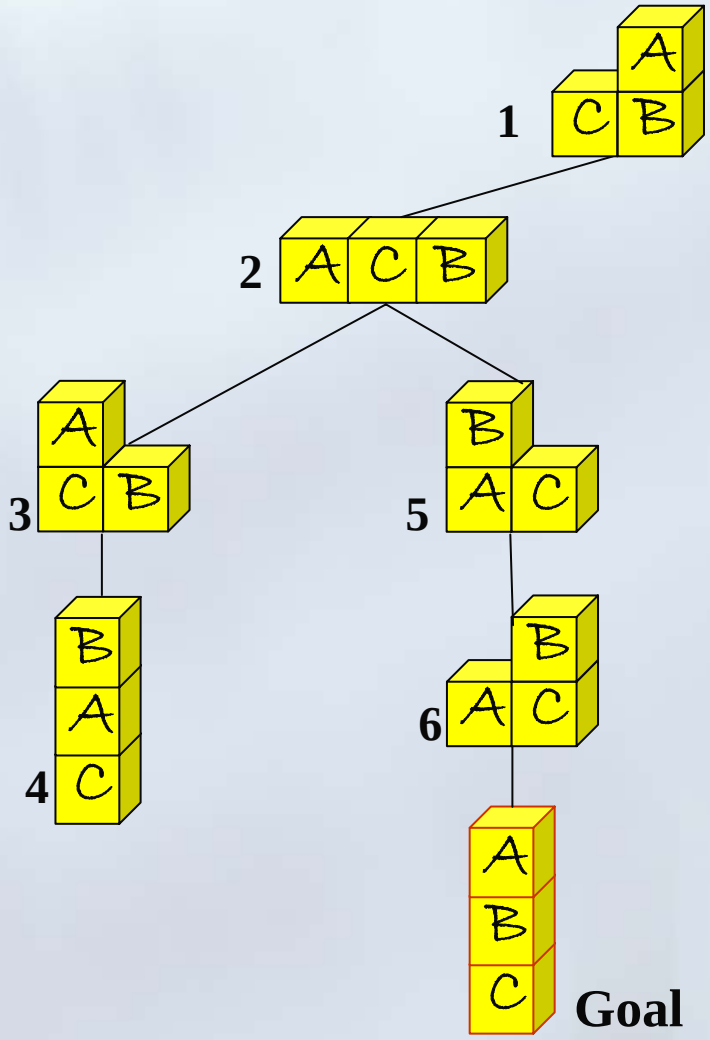


สถานะสุดท้าย

BFS : Block world



DFS : Block world



การค้นหาแบบฮิวริสติก : Heuristic search

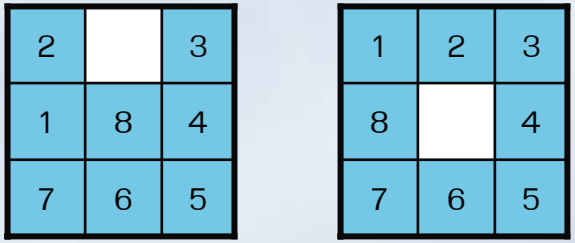
- เป็นวิธีการค้นหาที่นำความรู้เกี่ยวกับปัญหามาช่วยในการเลือกเส้นทาง
- เหมาะกับกรณีที่สเปซสถานะที่ต้องค้นหามีขนาดใหญ่
- ใช้การพิจารณาตัดโหนดที่ไม่จำเป็นต้องสำรวจออก
- ใช้แนวคิดว่ากระจายโหนดที่น่าจะไปถึงเป้าหมายก่อน
- ลดเวลาในการสำรวจ ไม่จำเป็นต้องกระจายทุกโหนด
- ความรู้ที่ใช้อยู่ในรูปฟังก์ชันฮิวริสติก

ฟังก์ชันฮิวริสติก : Heuristic function

- เป็นการคาดเดาอย่างมีเหตุผล อาจไม่ใช่ความรู้ที่สมบูรณ์
- ฟังก์ชันฮิวริสติกจะแปลงจากสถานะปัจจุบันไปเป็นตัวเลข
- ตัวเลขนี้เป็นตัวชี้บอกว่าสถานะนั้นเข้าใกล้เป้าหมายเพียงใด
- เป็นสิ่งที่ช่วยในการค้นหาว่าควรไปในทิศทางใด
- ถ้าฟังก์ชันดี ก็จะทำให้กระจายโหนดน้อย เข้าสู่เป้าหมายได้ไว
- ถ้าฟังก์ชันไม่ดี อาจทำให้ค้นหาผิด ได้คำตอบที่ไม่ใช่คำตอบที่ดีที่สุด หรือไม่เจอคำตอบได้
- โดยเฉลี่ยแล้ว จะค้นหาได้ดีกว่าการค้นหาแบบบอด

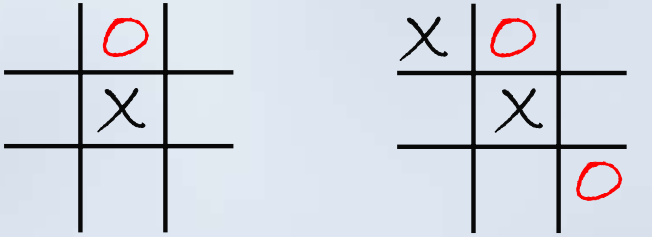
ตัวอย่างฟังก์ชันฮิวริสติก

- ปัญหาแปดปริศนา



- จำนวนตัวเลขที่อยู่ในตำแหน่งที่ถูกต้อง
- f(n) = 5,
  n คือ สถานะตั้งต้นดังรูป

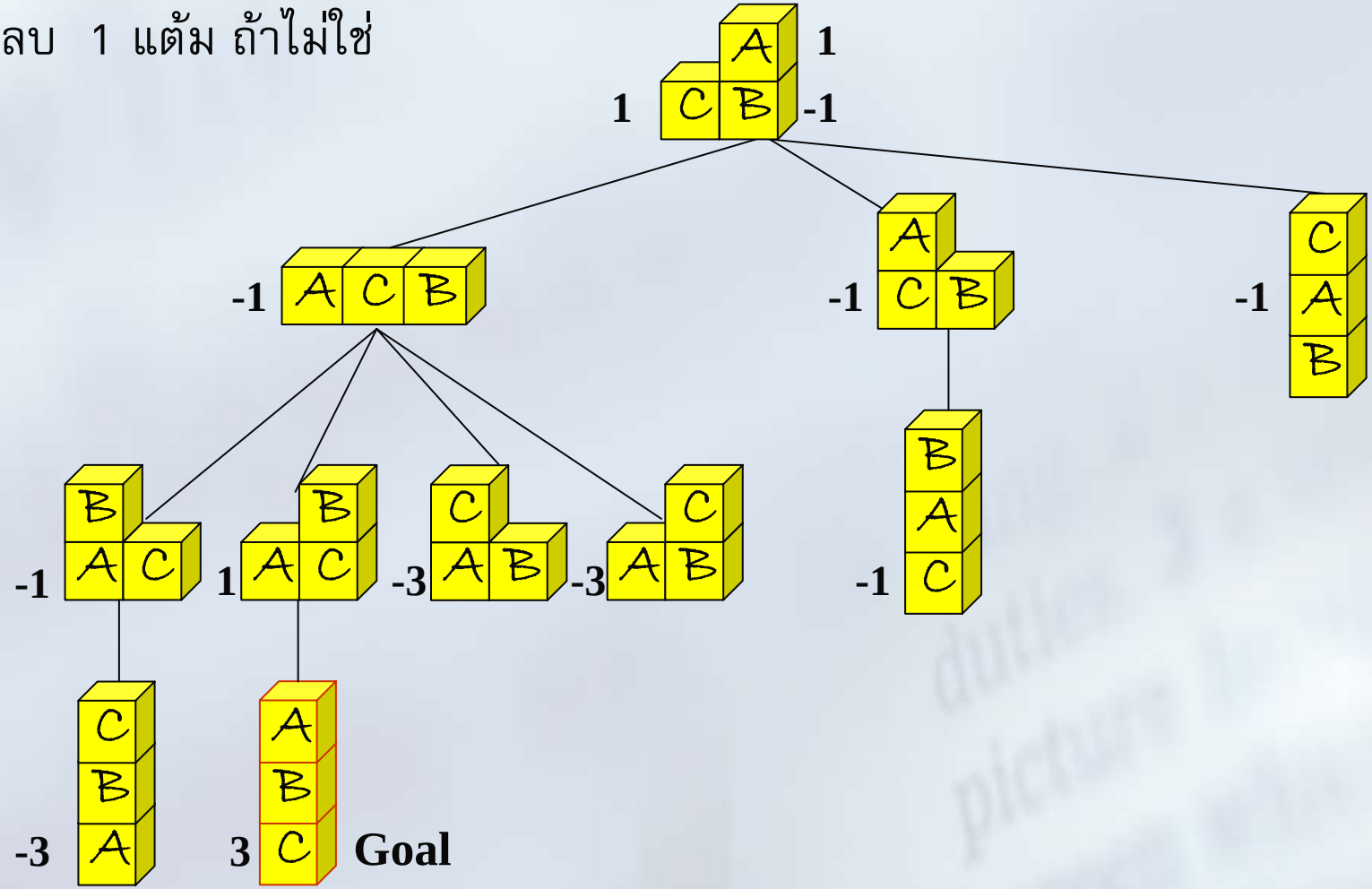
- เกมทิค-แทค-โท



- f(n) = จำนวนแนวเส้นตรงตามแนวนอน แนวตั้ง และแนวทแยงที่ **X** มีโอกาสกาชนะได้ - จำนวนแนวเส้นตรงตามแนวนอน แนวตั้ง และแนวทแยงที่ **O** มีโอกาสกาชนะได้

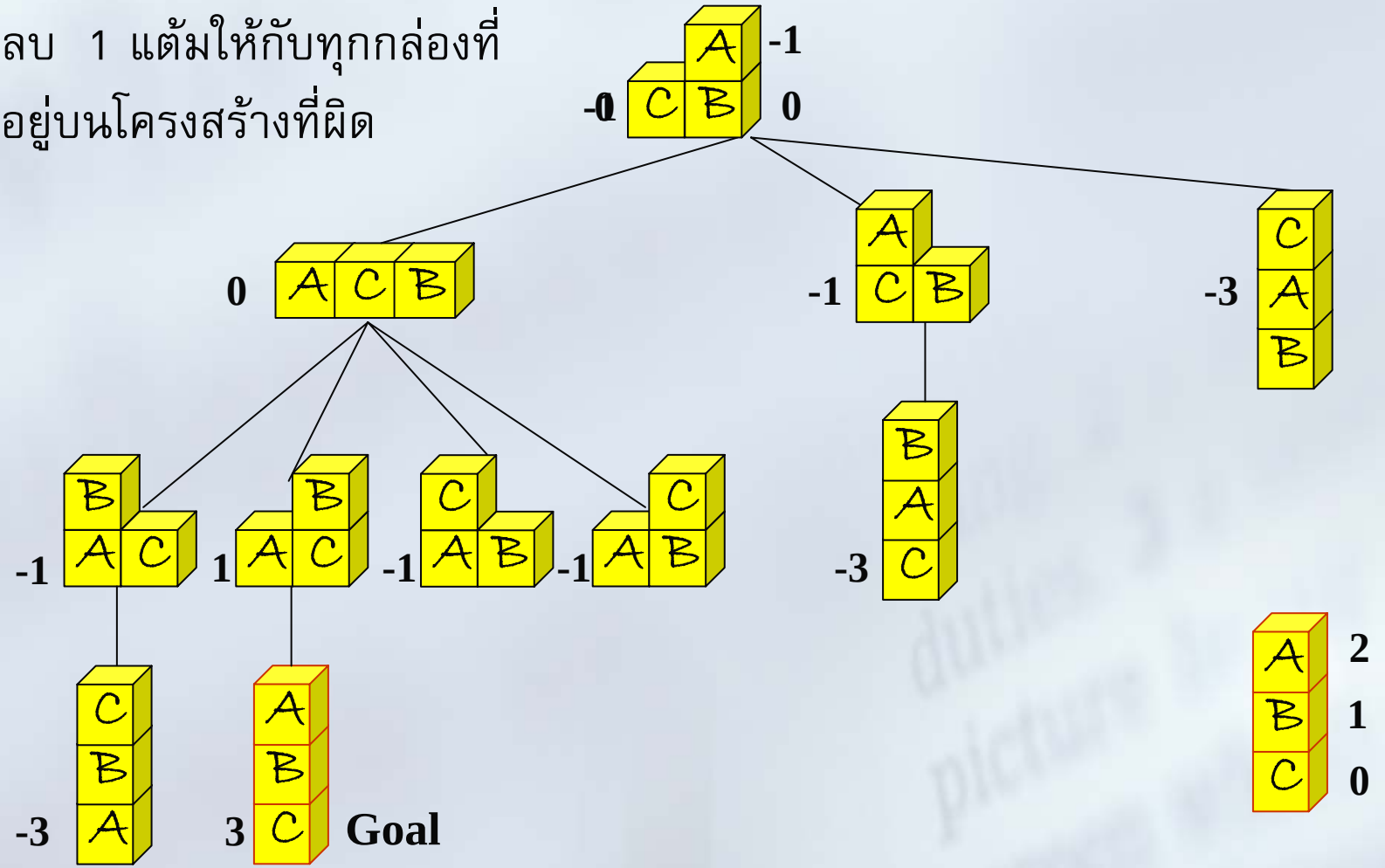
Heuristic function : block world #1

H1 : บวก 1 แต้มให้กับทุกกล่องที่วางอยู่บนสิ่งที่มันควรอยู่
ลบ 1 แต้ม ถ้าไม่ใช่

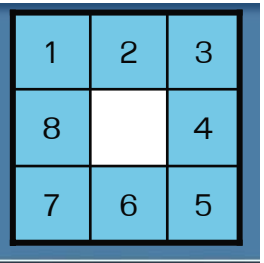


Heuristic function : block world #2

H2 : บวก 1 แต้มให้กับทุกกล่องที่อยู่บนโครงสร้างที่ถูก
ลบ 1 แต้มให้กับทุกกล่องที่
อยู่บนโครงสร้างที่ผิด



Heuristic function : 8-puzzle



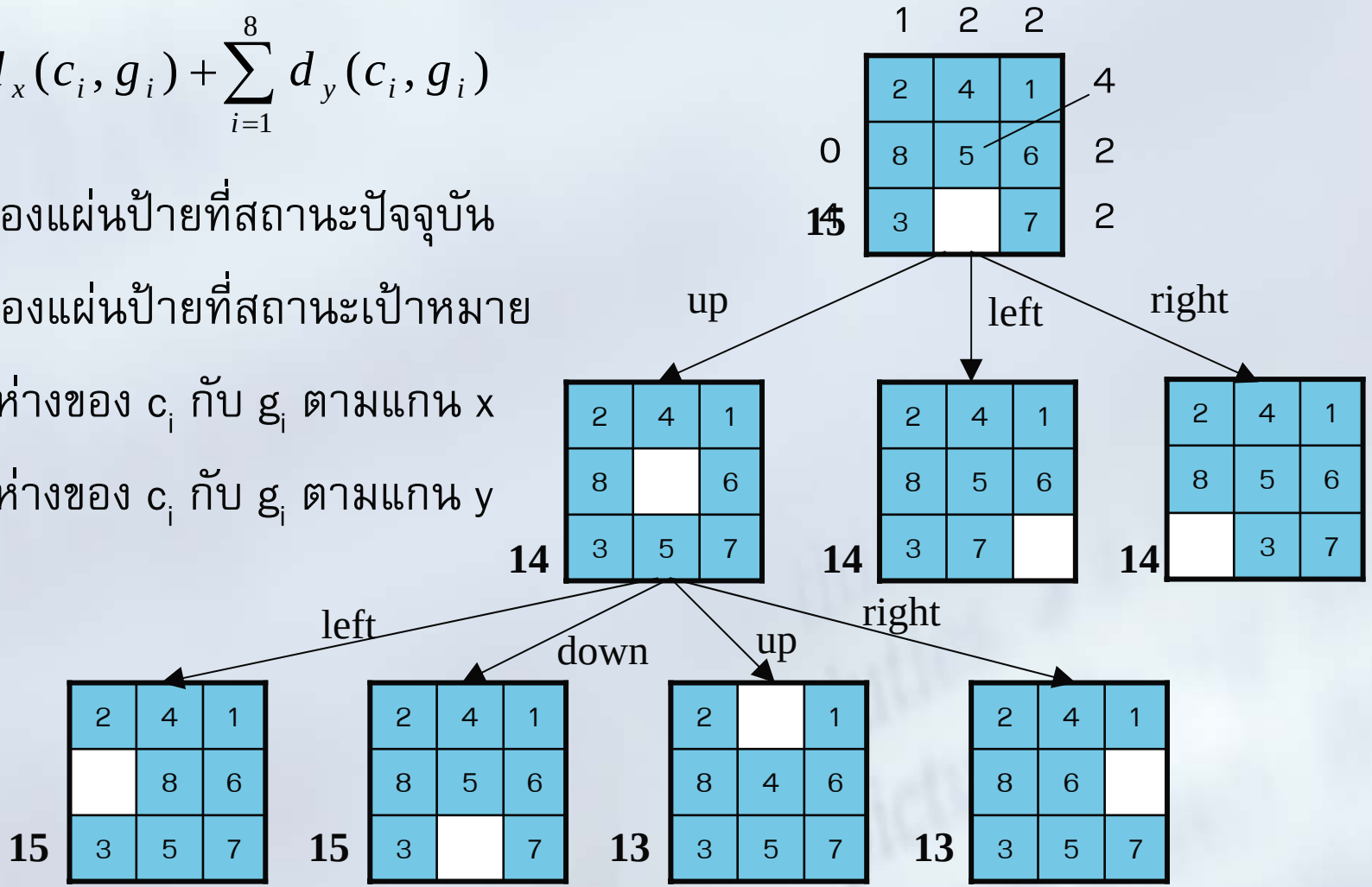
$$h = \sum_{i=1}^{8} d_x(c_i, g_i) + \sum_{i=1}^{8} d_y(c_i, g_i)$$

c_i = พิกัดของแผ่นป้ายที่สถานะปัจจุบัน

g_i = พิกัดของแผ่นป้ายที่สถานะเป้าหมาย

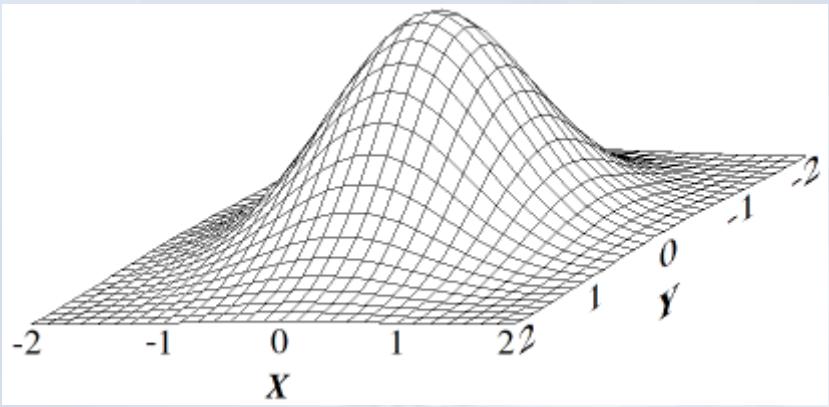
d_x = ระยะห่างของ c_i กับ g_i ตามแกน x

d_y = ระยะห่างของ c_i กับ g_i ตามแกน y



การค้นหาแบบปีนเขา : Hill climbing

- เป็นอัลกอริทึมที่นำฟังก์ชันฮิวริสติกมาช่วยค้นหาคำตอบ
- เปรียบการค้นหาเหมือนการปีนสู่ยอดเขา
- การขึ้นจะกระทำในแนวดิ่ง ถ้าเจอทางแยกก็เลือกทางที่ตรงดิ่งขึ้นเขา
- ค่าฮิวริสติก คือ ความสูงจากฐานถึงตำแหน่งปัจจุบัน
- ในกรณีนี้ค่ายิ่งสูงยิ่งดี
- มีสองแบบ
  - Simple
  - Steepest-ascent



Simple hill climbing algorithm

1. Evaluate initial state

   IF initial state = goal state THEN return initial state and quit.

   ELSE current state := initial state

2. UNTIL a goal state is found or no new operator left to be applied in the current state DO:

   2.1 select an operator that has not yet been applied to the current state and apply it to produce a new state

   2.2 Evaluate new state

   IF new state = goal state THEN return new state and quit

   ELSE IF h(new state) is better than h(current state)

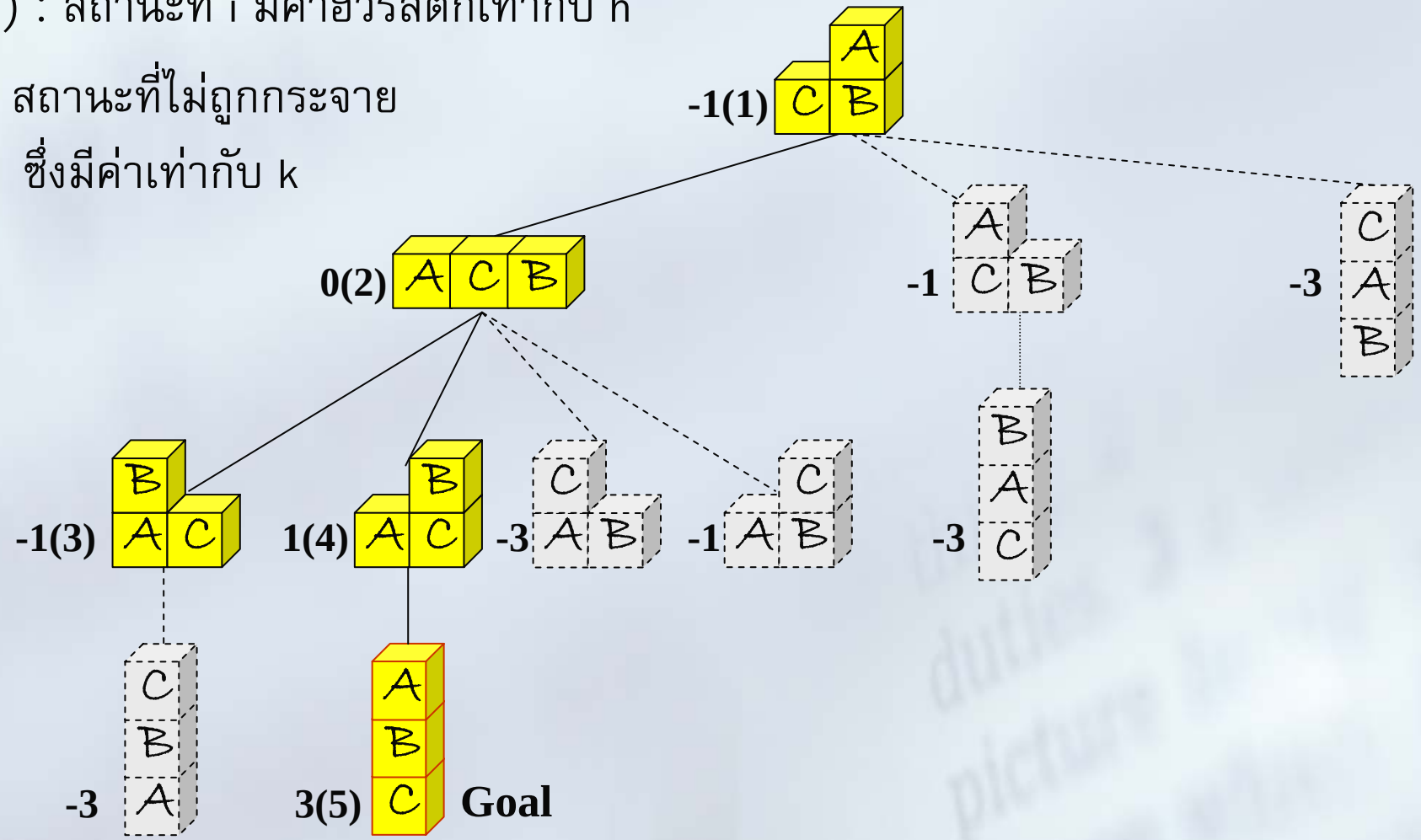
   THEN current state:= new state

   ELSE continue loop

Hill climbing : block world

h(i) : สถานะที่ i มีค่าฮิวริสติกเท่ากับ h

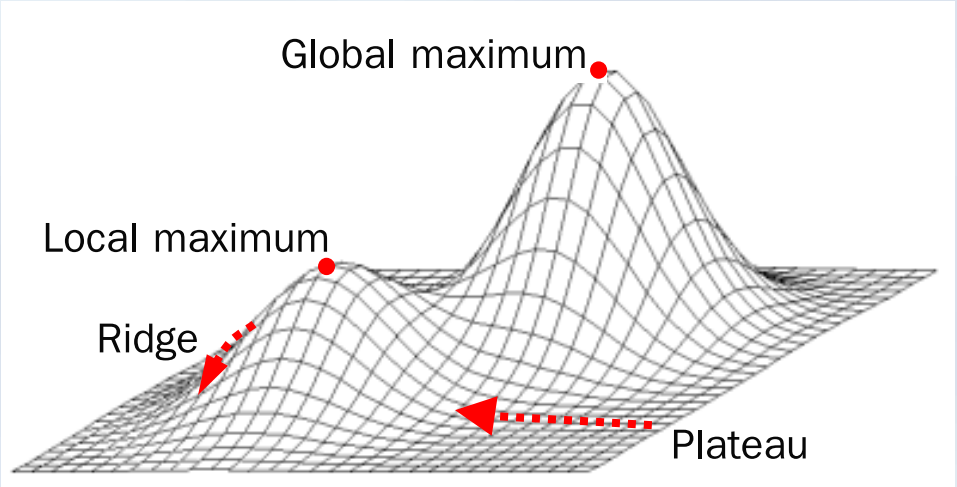
k : สถานะที่ไม่ถูกกระจาย
ซึ่งมีค่าเท่ากับ k



ปัญหาของการค้นหาแบบปีนเขา

ปัญหา

- Local maximum/foot hill
- Ridge
- Plateau

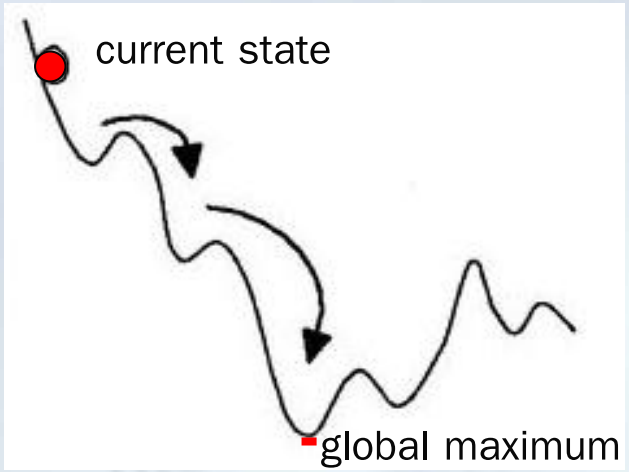


วิธีแก้ปัญหา

- ย้อนกลับไปตั้งต้นใหม่
- ย้ายไปตำแหน่งใหม่ที่ห่างจากจุดที่พบทางตัน
- ลองใช้ตัวกระทำการต่อเนื่องกันหลายๆ ตัว แล้วค่อยประเมินสถานะ

การค้นหาแบบจำลองการอบเหนียว : Simulated annealing

- เป็นอีกรูปแบบหนึ่งของ hill climbing ที่แก้ปัญหาเรื่อง local maximum
- ใช้แนวคิดอุณหพลศาสตร์ของกระบวนการอบเหนียวในฟิสิกส์
- เป็นกระบวนการหลอมละลายของโลหะ โดยลดอุณหภูมิลงอย่างช้าๆ ระหว่างหลอมเพื่อให้ได้โลหะที่อยู่ในสภาวะที่เหมาะสมที่สุด
- ออกแบบมาเพื่อให้หลุดจาก local maximum ได้ โดยยอม search ไปในทางที่ไม่ดีใช้ช่วงเริ่มต้นของการค้นหา



การจำลองการอบเหนียว

- เปรียบสถานะปัจจุบันของปัญหาเสมือนสถานะปัจจุบันของโครงสร้างโลหะ
- เปรียบค่าฮิวริสติกเสมือนพลังงานของโครงสร้างโลหะ
- เปรียบการวนลูปเสมือนการค่อยๆ ลดอุณหภูมิลงเรื่อยๆ
- การสร้างเส้นทางใหม่โดยปรับเปลี่ยนเส้นทางปัจจุบันเปรียบเสมือนการปรับเปลี่ยนโครงสร้างโลหะระหว่างการอบเหนียว
- การยอมรับโครงสร้างใหม่ ขึ้นกับความน่าจะเป็น
  - ความน่าจะเป็นที่จะยอมรับโครงสร้างใหม่

$$p = e^{-\Delta E / kT}$$

ΔE คือ ระดับพลังงานที่เปลี่ยนไป

T คือ อุณหภูมิ

k คือ ค่าคงที่

  - ความน่าจะเป็นที่จะค้นหาในทิศทางที่ไม่ดี

$$p = e^{-\Delta E / kT}$$

ΔE คือ ค่าฮิวริสติกที่เปลี่ยนไป

T คือ อุณหภูมิ (ค่อยๆ ลดลง)

Simulated annealing algorithm

1. Evaluate initial state

   IF initial state = goal state

   THEN return initial state and quit

   ELSE current state := initial state

2. *BEST-SO-FAR* := current state

3. *T* := constant

4. UNTIL found solution or no new operator left to be applied in the current state do:

   4.1 select an operator that has not yet been applied to the current state and apply it to produce a new state

   4.2 Evaluate new state

   dE := h(current state)-h(new state)

   IF new state = goal state

   THEN return new state and quit

   ELSE IF h(new state) is better than h(current state)

   THEN current state := new state

   IF h(new state) is better than h(BEST-SO-FAR) THEN

   BEST-SO-FAR := new state

   ELSE compute ***p***

   IF ***p*** > random(0,1)

   THEN current state := new state

   4.3 Revise *T* as necessary

5. Return *BEST-SO-FAR* as the answer

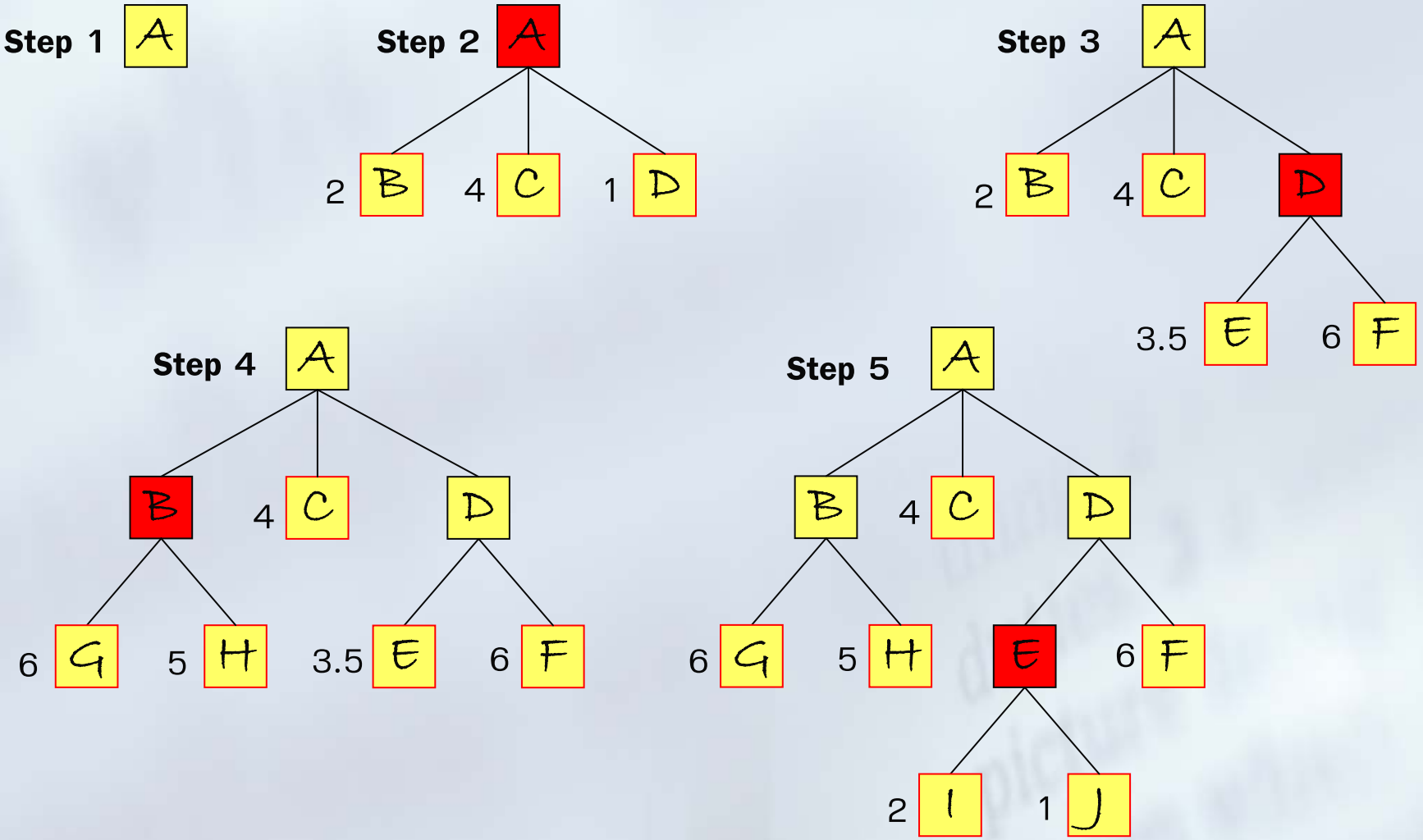
การค้นหาแบบดีที่สุดก่อน : Best-first search

- เป็นการนำข้อดีของ BFS และ DFS มารวมกัน แก้ปัญหา local optimum
- คล้ายกับการค้นหาแบบปีนเขา มีข้อแตกต่างคือ
  - แบบปีนเขา จะเลือกเส้นทางแล้วตัดตัวเลือกอื่นที่เป็นลูกของสถานะปัจจุบันทิ้ง
  - แบบดีที่สุดก่อน จะเก็บสถานะทุกตัวไว้โดยไม่มีการตัดทิ้ง
- เลือกสถานะที่มีค่าฮิวริสติกที่ดีที่สุด โดยพิจารณาจากสถานะทุกตัวที่ยังไม่ถูกกระจาย
- สร้างสถานะลูกของตัวที่ถูกเลือก ถ้ายังไม่พบคำตอบ นำสถานะลูกเหล่านี้เปรียบเทียบกับสถานะที่เก็บไว้และยังไม่ได้เลือก
- แบบปีนเขาเปรียบเทียบเฉพาะสถานะลูกด้วยกันเอง
- ไม่พลาดเส้นทางที่จะนำไปสู่คำตอบ

Best-first search algorithm

1. *OPEN* := {initial state}
2. UNTIL a gold is found or no nodes left on *OPEN* DO:
   - 2.1 Pick the best node on *OPEN*
   - 2.2 Generate its successors
   - 2.3 FOR each successor DO:
     - 2.3.1 IF it has not been generated before, evaluate it, add it to *OPEN*, and record its parent
     - 2.3.2 IF it has been generated before, IF this path is better than the previous one THEN change the parent and update cost

Best-first search



การค้นหาแบบเอ-สตาร์ : A* algorithm

- เป็นรูปแบบหนึ่งของการค้นหาแบบดีที่สุดก่อน
- ฟังก์ชันฮิวริสติก พิจารณาต้นทุนจากสถานะเริ่มต้นมายังสถานะปัจจุบันและจากสถานะปัจจุบันไปยังสถานะเป้าหมาย
- ใช้วิธีประมาณการ หาระยะทางจากสถานะปัจจุบันไปสถานะเป้าหมาย

$$f'(s) = g(s) + h'(s)$$

- โดยที่ g คือ ฟังก์ชันที่คำนวณ cost จากสถานะเริ่มต้นไปยังสถานะปัจจุบัน

  h' คือ ฟังก์ชันที่ประมาณ cost จากสถานะปัจจุบันไปยังสถานะเป้าหมาย

  f' คือ ฟังก์ชันที่ประมาณ cost จากสถานะเริ่มต้นไปยังสถานะเป้าหมาย

  (ยิ่งค่าน้อยยิ่งดี)

ข้อดีของ A* algorithm

- อัลกอริทึม A* จะเหมือนกับ Best-first ทุกอย่าง ยกเว้นฟังก์ชันฮิวริสติกที่ใช้ f' คือ พิจารณา cost ทั้งหมด
- ถ้ามีคำตอบ A* จะนำไปสู่คำตอบที่ดีที่สุด (cost น้อยสุด) เสมอ
- การใช้ค่า g อาจทำให้สถานะน่าจะเข้าใกล้สถานะเป้าหมายมากที่สุดไม่ถูกกระจายได้
- ถ้าให้ g ของทุกสถานะเป็น 0 ก็คือ Best-first search นั่นเอง

$$f'(s) = g(s) + h'(s)$$

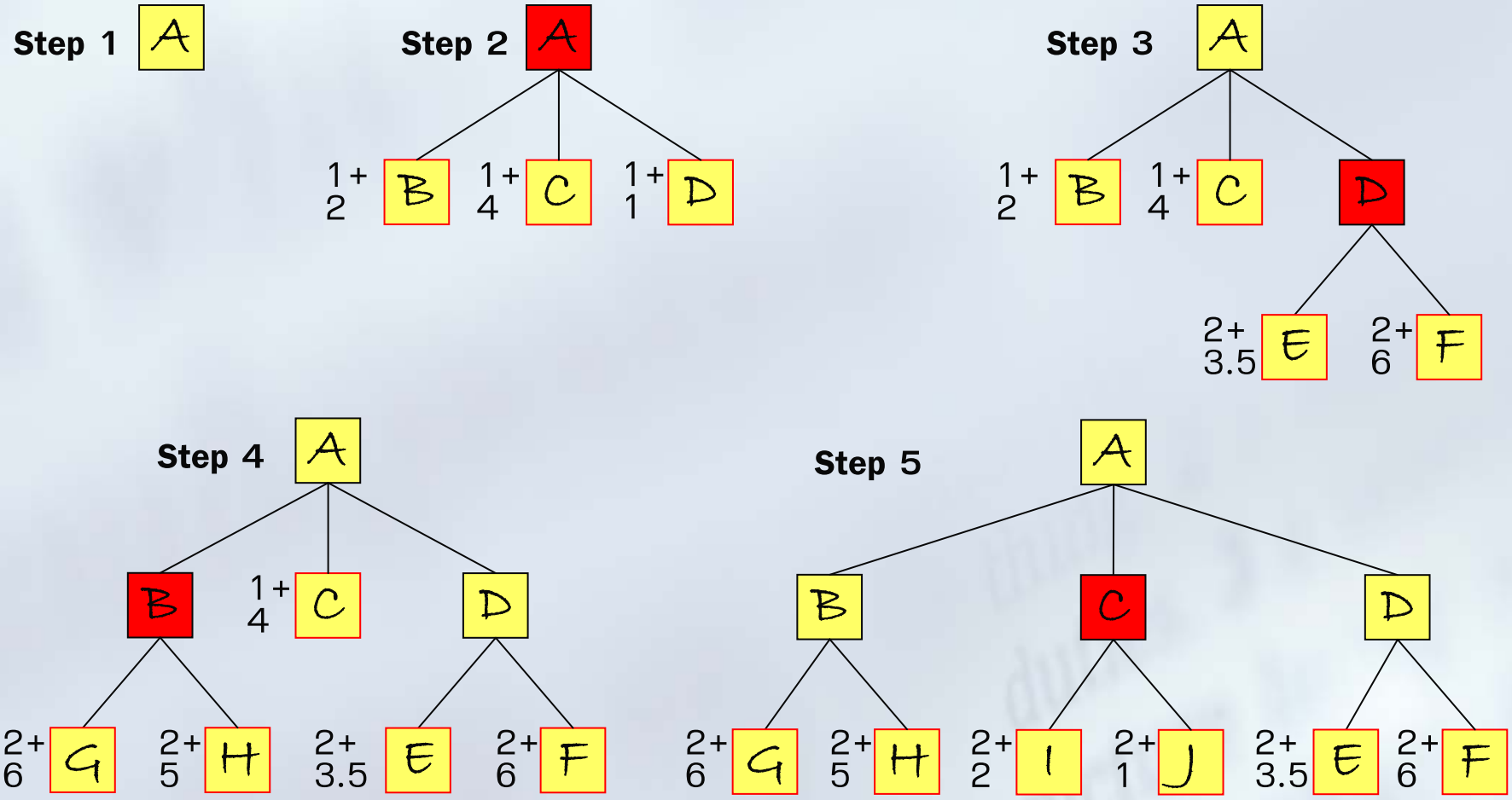
0

- ถ้าให้ g ของทุกสถานะเป็น 1 จะได้คำตอบที่ให้ step น้อยที่สุด

$$f'(s) = g(s) + h'(s)$$

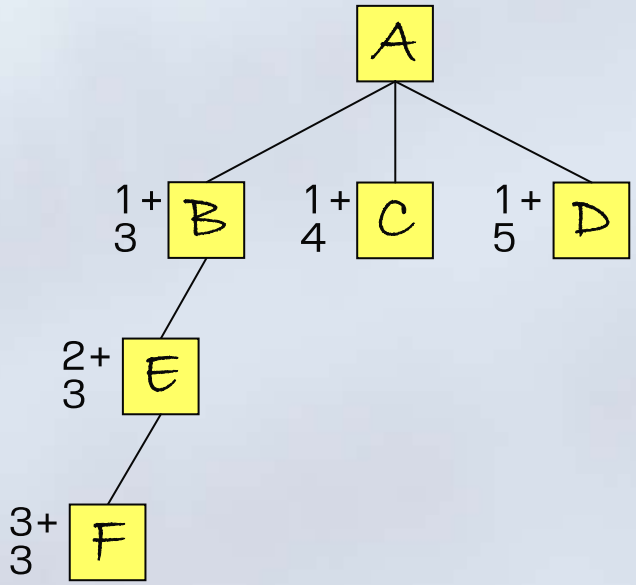
1

A* search

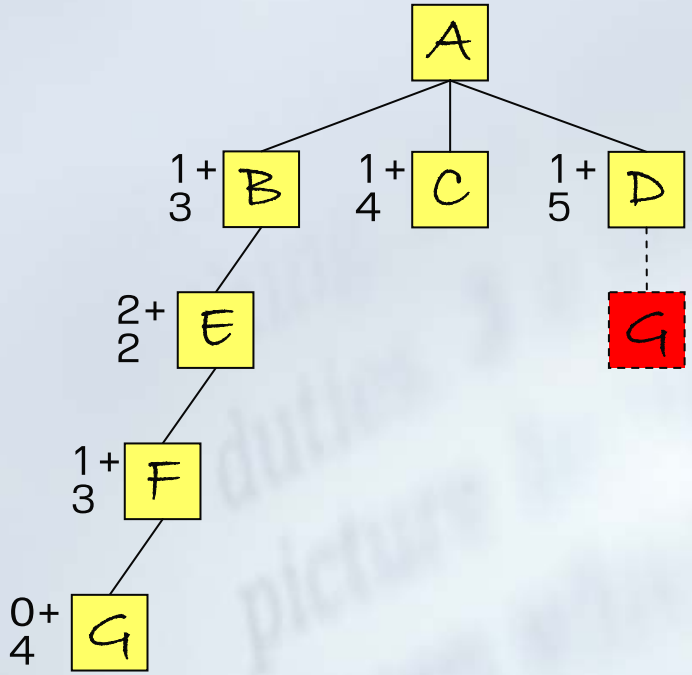

Step 1
A
Step 2
A
1+2 B
1+4 C
1+1 D
Step 3
A
1+2 B
1+4 C
D
2+3.5 E
2+6 F
Step 4
A
B
1+4 C
D
2+6 G
2+5 H
2+3.5 E
2+6 F
Step 5
A
B
C
D
2+6 G
2+5 H
2+2 I
2+1 J
2+3.5 E
2+6 F

A* algorithm : under-estimate VS over-estimate

- ถ้า h' เป็นฟังก์ชันที่ประมาณค่าน้อยกว่าความเป็นจริงเสมอ (under-estimating function) ของ h จะประกันได้ว่าเส้นทางที่ได้เป็นเส้นทางที่ดีที่สุด (optimal path)



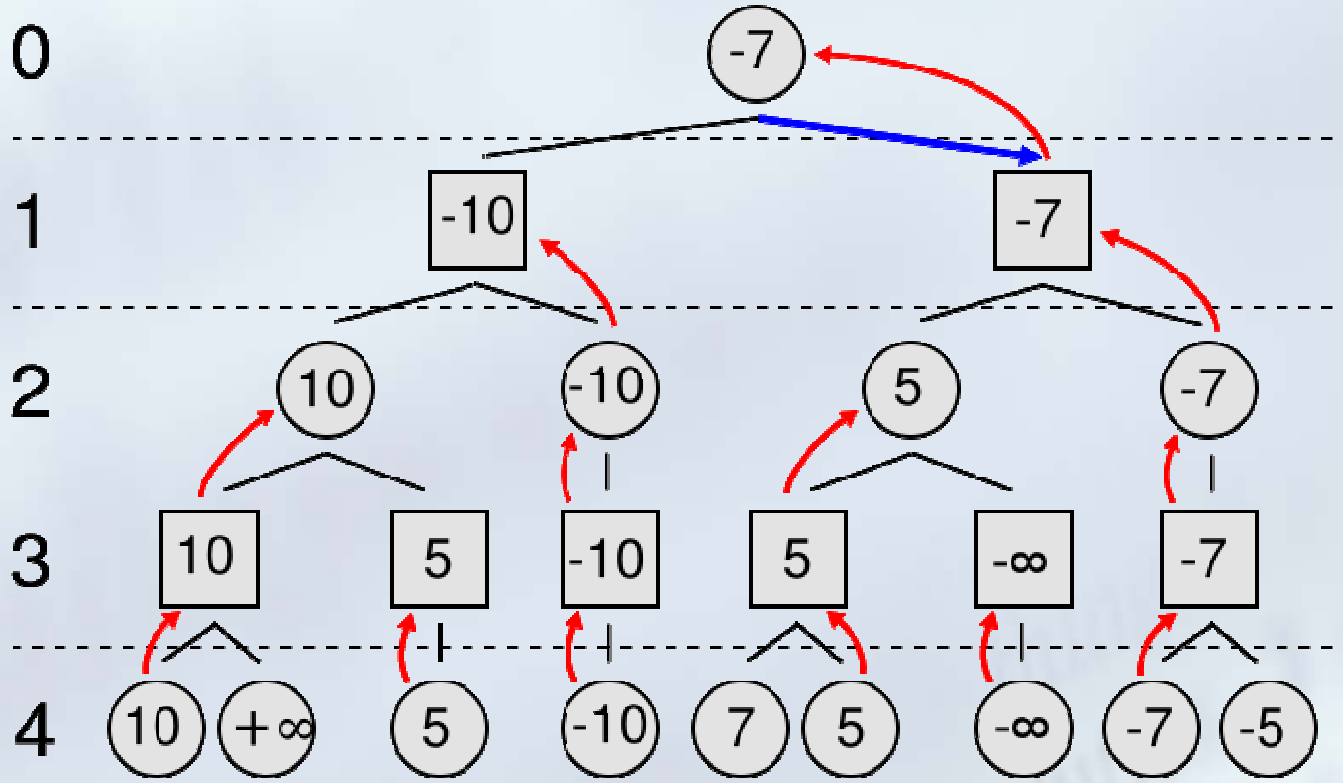
- ถ้า h' เป็นฟังก์ชันที่ประมาณค่ามากกว่าความเป็นจริงเสมอ (over-estimating function) ของ h อาจจะทำให้การค้นหาผิดทางได้



การค้นหาคำตอบสำหรับต้นไม้เกม

- เป็นการค้นหาที่เหมาะกับการเล่นเกมต่างๆ ที่มักจะมีผู้เล่น 2 ฝ่าย
- แทนปัญหาในรูปสเปซของสถานะ
- การค้นหาต้นไม้เกม (game tree) ใช้แทนสเปซของสถานะที่เป็นไปได้ของเกมนั้นๆ
- หาเส้นทางที่จะทำให้เล่นชนะคู่ต่อสู้ได้
- ตัวอย่างวิธีการค้นหาต้นไม้เกม
  - การค้นหาแบบมินิแมกซ์ (Minimax)
  - การค้นหาแบบอัลฟา-เบตา (Alpha-beta)
  - การค้นหาแบบฮิวริสติกโดยตัดเส้นทางทิ้ง (Heuristic pruning)

Minimax algorithm



represent the moves of the player running the algorithm (*maximizing player*)

represent the moves of the opponent (*minimizing player*)