ปัญหาและการนิยามปัญหา

ทัศนวรรณ ศูนย์กลาง

เนื้อหาการบรรยาย

- ปัญหา
- การนิยามปัญหา
- ตัวอย่างการนิยามปัญหา
- การแก้ปัญหาทางปัญหาประดิษฐ์
- การวิเคราะห์ปัญหา
- การค้นหาคำตอบ
- เทคนิคการค้นหา

ปัญหา : Problems

- เอไอ คือ การสร้างคอมพิวเตอร์ให้มี “ปัญญา”
- สร้างคอมพิวเตอร์ให้รู้จักการแก้ปัญหา
- ปัญหาทางเอไอมีความหลากหลาย มีลักษณะที่แตกต่างกันไป
- ปัญหามีความซับซ้อนและมีความไม่แน่นอน
- การแก้ปัญหาจึงต้องอาศัย “ความรู้” เป็นหลัก
- ความรู้มีคุณสมบัติที่ยากต่อการประมวลผล
- มีปริมาณมาก ยากที่จะบอกลักษณะได้อย่างถูกต้อง มีการเปลี่ยนแปลงอยู่ตลอด การจัดเก็บต้องเหมาะแก่การนำไปใช้

การแทนความรู้ : Knowledge representation

- การแก้ปัญหาทางเอไอจึงเป็นวิธีการที่ใช้ความรู้
- การแทนความรู้ที่ใช้ในเอไอต้องอยู่ในรูปแบบที่เหมาะสมต่อการจัดเก็บและนำไปใช้ได้ง่าย
- ควรมีลักษณะต่อไปนี้
  - มีความเป็นสากล (Generalizations)
  - เป็นรูปแบบที่ผู้ให้ความรู้สามารถเข้าใจได้
  - สามารถแก้ไขข้อผิดพลาดและเปลี่ยนแปลงให้ทันสมัยได้
  - สามารถนำไปใช้ได้กับหลายสถานการณ์ แม้ว่าความรู้นั้นจะไม่ถูกต้องหรือสมบูรณ์
  - สามารถช่วยลดขอบเขตของคำตอบที่เป็นไปได้ให้แคบลง

การนิยามปัญหา : Defining the problem

- การนิยามปัญหาถึงเป็นหัวใจสำคัญในการเริ่มต้นแก้ปัญหา
- ขั้นตอนที่สำคัญในการแก้ปัญหาหนึ่งๆ มีดังนี้
  - **นิยามปัญหาให้ชัดเจน** ระบุสถานะต่างๆ ของปัญหา
  - **วิเคราะห์ปัญหา** มีลักษณะสำคัญอะไรบ้าง
  - **ค้นหาและแทนความรู้ที่ใช้แก้ปัญหา** มีความรู้แบบใดบ้างที่ต้องนำมาใช้
  - **เลือกวิธีการแก้ปัญหา** เลือกวิธีที่คิดว่าเหมาะสม
- การนิยามปัญหานิยมแทนให้อยู่ในรูปสเปซของสถานะต่างๆ ของปัญหา (state space)

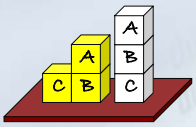
สเปซสถานะ : State space

- เป็นการแทนปัญหาในรูปของสเปซของสถานะต่างๆ
- การนิยามปัญหากระทำโดยระบุ
  - **สเปซสถานะ (State space)** โดยแสดงวัตถุทั้งหมดที่เกี่ยวข้องกับปัญหา
  - **สถานะเริ่มต้น (Initial state)** แทนตำแหน่งเริ่มต้นของวัตถุทั้งหมด
  - **สถานะเป้าหมาย (Goal/final state)** แทนตำแหน่งสุดท้ายซึ่งเป็นคำตอบ
  - **สถานะที่เป็นไปได้ทั้งหมด (All possible states)** มักมีจำนวนมาก
  - **ตัวกระทำการ (Operators)** กฎที่ใช้เปลี่ยนสถานะหนึ่งไปเป็นอีกสถานะหนึ่ง
  - **เงื่อนไข/ข้อจำกัด (Path constraint)** ในการไปสู่สถานะเป้าหมาย

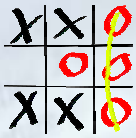
คำอธิบายสถานะ : Description of the state

- สถานะของปัญหา คือ สภาพของปัญหาที่อาจเกิดขึ้นหรือเป็นไปได้
- การแทนสถานะในคอมพิวเตอร์อาจทำได้หลายแบบ
- เมื่อแทนแล้วต้องช่วยให้แก้ปัญหาได้ง่าย
- ไม่จำเป็นต้องอธิบายสถานะให้เหมือนสถานะจริงทุกประการ

8-puzzle



Block world



Tic-tac-toe

State space : 8-puzzle problem

- คู่ลำดับ (X,Y,Z) ของตำแหน่งและตัวเลข
  X แทนแถว, Y แทนคอลัมน์, Z แทนตัวเลขในตำแหน่งนั้น
  (1,1,4)
  (3,3,7)
- เมตริกซ์ขนาด 3x3 (อาร์เรย์ 2 มิติ)
  { 4, 3, 6,
  5, 8, 0,
  1, 2, 7 }

4	3	6
5	8	
1	2	7

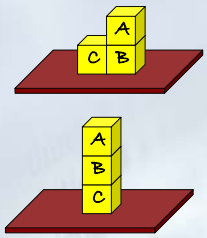
1	2	3
4	5	6
7	8	

State space : water jug problem

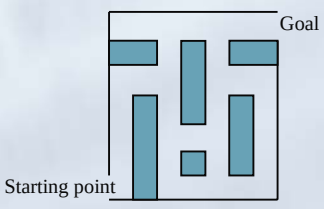
- คู่ลำดับ (X, Y) โดยที่
  X แทนความจุน้ำในถัง 8 แกลลอน
  Y แทนความจุน้ำในถัง 6 แกลลอน
- X = 0, 1, 2, 3, 4, 5, 6, 7, 8
- Y = 0, 1, 2, 3, 4, 5, 6
- (0,0)
- (0,4)

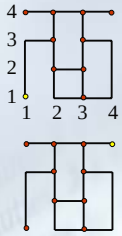
State space : block world problem

- ความสัมพันธ์ระหว่างโต๊ะและกล่อง
- X และ Y แทนกล่อง
- TABLE แทนโต๊ะ
- ON(X,Y) แทน X อยู่บน Y
- CLEAR(X) แทนไม่มีกล่องอื่นวางบน X
- ON(A, B)
- ON(C, TABLE)
- CLEAR(C)



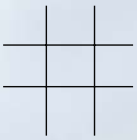
State space : maze

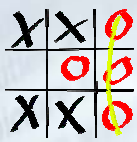




State space : tic-tac-toe

- เมตริกซ์ขนาด 3x3
- ผู้เล่น 2 คน ผลัดกันกาเครื่องหมาย
- ตำแหน่งของเครื่องหมายบนกระดานระบุว่าใครเป็นผู้เล่น
- ถ้ากาเครื่องหมายเป็นเส้นตรงทิศใดก็ได้จะถึงว่าชนะ
- เกมสิ้นสุดเมื่อฝ่ายหนึ่งชนะหรือกาเครื่องหมายจนเต็ม





ปัญหาแปดปริศนา : 8-puzzle problem

- ประกอบด้วยช่องว่าง 9 ช่อง
- ตัวเลข 1-8 บรรจุอยู่ในช่อง
- มีช่องว่างหนึ่งช่อง เลื่อนเลขไปมาได้
- เป้าหมายคือเรียงตัวเลขในช่องให้เรียงจากน้อยไปมาก ตามเข็มนาฬิกา
- สถานะเริ่มต้น {2,0,3,1,5,4,7,6,5}
- สถานะสุดท้าย {1,2,3,8,0,4,7,6,5}
- สถานะที่เป็นไปได้ = 9!
- สถานะเปลี่ยนเมื่อมีการเลื่อนตัวเลขแทนที่ช่องว่าง
- ไม่มีเงื่อนไขในการไปสู่เป้าหมาย

สถานะเริ่มต้น

2		3
1	8	4
7	6	5

สถานะสุดท้าย

1	2	3
8		4
7	6	5

ตัวกระทำการของแปดปริศนา

เงื่อนไขเบื้องต้น	ตัวกระทำการ	
o ด้านบนของแผ่นป้ายมีช่องว่าง	เลื่อนแผ่นป้ายขึ้นบน	ยึดแผ่นป้ายเป็นหลัก
o ด้านล่างของแผ่นป้ายมีช่องว่าง	เลื่อนแผ่นป้ายลงข้างล่าง	
o ด้านซ้ายของแผ่นป้ายมีช่องว่าง	เลื่อนแผ่นป้ายไปทางซ้าย	
o ด้านขวาของแผ่นป้ายมีช่องว่าง	เลื่อนแผ่นป้ายไปทางขวา	

เงื่อนไขเบื้องต้น	ตัวกระทำการ	
• ด้านบนช่องว่างมีแผ่นป้าย	สลับตำแหน่งช่องว่างกับแผ่นป้าย	ยึดช่องว่างเป็นหลัก
• ด้านล่างช่องว่างมีแผ่นป้าย	สลับตำแหน่งช่องว่างกับแผ่นป้าย	
• ด้านซ้ายช่องว่างมีแผ่นป้าย	สลับตำแหน่งช่องว่างกับแผ่นป้าย	
• ด้านขวาช่องว่างมีแผ่นป้าย	สลับตำแหน่งช่องว่างกับแผ่นป้าย	

ปัญหาเหยือกน้ำ : Water jug problem

- มีถังน้ำความจุ 6 และ 8 แกลลอน
- ไม่มีมาตรวัดบอกปริมาตรน้ำ
- ต้องการให้ถังความจุ 8 แกลลอน บรรจุน้ำ 4 แกลลอน
- สถานะเริ่มต้น (0,0)
- สถานะสุดท้าย (4,n)
- สถานะทั้งหมด = เซ็ตของ (x,y)
- สถานะเปลี่ยนเมื่อมีการเทน้ำออกหรือเติมน้ำลงในแกลลอน
- ไม่มีเงื่อนไขในการไปสู่เป้าหมาย

ตัวกระทำการของปัญหาเหยือกน้ำ

- ตัวกระทำการ มีดังนี้

1	เติมน้ำถัง 8 ให้เต็ม	(x, y | x < 8) -> (8, y)
2	เติมน้ำถัง 6 ให้เต็ม	(x, y | y < 6) -> (x, 6)
3	เทน้ำถัง 8 ออกหมด	(x, y | x > 0) -> (0, y)
4	เทน้ำถัง 6 ออกหมด	(x, y | y > 0) -> (x, 0)
5	เทน้ำถัง 8 ทั้งหมดใส่ถัง 6	(x, y | x+y <= 6, x > 0) -> (0, x+y)
6	เทน้ำถัง 6 ทั้งหมดใส่ถัง 8	(x, y | x+y <= 8, y > 0) -> (x+y, 0)
7	เทน้ำถัง 8 ใส่ถัง 6 จนถัง 6 เต็ม	(x, y | x+y >= 6, x > 0) -> (x-(6-y), 6)
8	เทน้ำถัง 6 ใส่ถัง 8 จนถัง 8 เต็ม	(x, y | x+y <= 8, y > 0) -> (8, y-(8-x))

การแก้ปัญหา : Problem solving

- **แบบที่ 1** วิเคราะห์ว่าต้องใช้ตัวกระทำการใดบ้าง จึงจะทำให้ได้น้ำ 4 แกลลอนบรรจุอยู่ในถัง 8 แกลลอน
  - ลำดับของตัวกระทำการ 2, 6, 2, 8, 3, 6
  - เขียนโปรแกรมโดยระบุลำดับตัวกระทำการนี้
  - วิธีการแบบกำหนดวิธีแก้ปัญหาไว้เรียบร้อยแล้ว
  - ทำได้งานเร็ว มีประสิทธิภาพ สร้างได้ง่าย
- **แบบที่ 2** แทนปัญหาให้อยู่ในรูปแบบสากล และเขียนโปรแกรมให้คอมพิวเตอร์ค้นหาคำตอบเอง
  - ให้ (x, y) แทนปริมาณน้ำในถัง 8 และ 6 แกลลอน ตามลำดับ
  - ใช้เทคนิคการค้นหาในการเปลี่ยนจากสถานะเริ่มต้น (0,0) โดยนำตัวกระทำการมาใช้ แล้วตรวจสอบสถานะจนกว่าจะได้สถานะที่ต้องการ (4,n)

วิธีการแก้ปัญหาทางปัญญาประดิษฐ์

- การใช้วิธีการค้นหาเส้นทางที่นำสู่สถานะเป้าหมาย
- ใช้วิธีการนิยามปัญหาในรูปของสถานะต่างๆ และตัวกระทำการ
- อาจนำความรู้มาช่วยให้การค้นหาคำตอบให้เร็วขึ้น
- สามารถใช้วิธีการแก้ปัญหาที่แตกต่างกันได้
- นำกระบวนการคิดหาคำตอบ (reasoning process) ใส่ในโปรแกรม
- จัดเก็บความรู้เกี่ยวกับปัญหาแยกกับกระบวนการคิดหาคำตอบได้
- ง่ายต่อการปรับปรุง แก้ไขความรู้
- จากตัวอย่าง คือ วิธีการแบบที่ 2

การวิเคราะห์ปัญหา : Analyze the problem

- วิธีการหาคำตอบมีหลายวิธี เหมาะกับปัญหาที่มีลักษณะต่างกันไป
- ต้องเข้าใจลักษณะของปัญหาให้ดี จึงจะเลือกใช้วิธีได้เหมาะสม
- ลักษณะของปัญหาที่ควรพิจารณาในหลายๆ แง่มุม เช่น
  - ปัญหาสามารถแยกเป็นปัญหาย่อยที่เป็นอิสระต่อกันได้หรือไม่
  - ขั้นตอนในการแก้ปัญหาสามารถย้อนกลับหรือยกเลิกได้หรือไม่ ถ้าทำไปแล้วและพบว่าไม่ดี
  - ปัญหามีความไม่แน่นอนเข้ามาเกี่ยวข้องด้วยหรือไม่
  - คำตอบของปัญหาสามารถรู้ได้โดยไม่ต้องเปรียบเทียบกับคำตอบอื่นๆ หรือไม่
  - ฐานความรู้ที่ใช้ในการแก้ปัญหามีความสอดคล้องกันหรือไม่
  - จำเป็นต้องใช้ความรู้จำนวนมากในการแก้ปัญหา หรือจำเป็นต้องใช้เพื่อจำกัดขอบเขตการค้นหาหรือไม่
  - จำเป็นต้องใช้คนช่วยในระหว่างขั้นตอนการแก้ปัญหาหรือไม่ (human interaction)

แยกเป็นปัญหาย่อยได้ : Decomposable

ตัวอย่างปัญหาที่แยกได้

- การหาค่าของการอินทิเกรต

$$\int x^2 + 3x + \sin^2 x \cos^2 x dx$$

- สามารถแยกได้เป็น 3 ปัญหาย่อย

$$\int x^2 dx$$

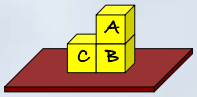
$$\int 3x dx$$

$$\int \sin^2 x \cos^2 x dx$$

- คำตอบได้จากคำตอบย่อยทั้งสาม

ตัวอย่างปัญหาที่แยกไม่ได้

- ปัญหาของ block world



- เป้าหมายคือ วางกล่อง A ทับกล่อง B และกล่อง B ทับกล่อง C

  ON(B,C) ^ ON(A,B)
- คำตอบของสองปัญหาย่อยนี้ นำมารวมแล้วไม่ใช่คำตอบของปัญหาใหญ่

ย้อนกลับหรือยกเลิกขั้นตอนได้ : Undone/ignore

ตัวอย่างปัญหาที่ยกเลิกได้

- การพิสูจน์ทฤษฎีทางคณิตศาสตร์
- เริ่มด้วยการพิสูจน์ทฤษฎีย่อยๆ ก่อน
- ถ้าพบว่าไม่มีประโยชน์ก็ยกเลิกทฤษฎีย่อยเหล่านี้ได้ ไม่จำเป็นต้องแก้ไขหรือทำอะไร

ตัวอย่างปัญหาที่ย้อนกลับได้

- ปัญหาแปดปริศนา

- เมื่อเลื่อน 3 ไปแทนที่ว่างแล้วคิดว่าน่าจะเลื่อน 2 ลงมาแทนดีกว่า
- จะเปลี่ยนโดยเลื่อน 2 ลงมาเลยไม่ได้ ต้องเลื่อน 3 กลับไปที่เดิมก่อน จึงจะเลื่อน 2 ลงมาได้

มีความไม่แน่นอนเข้ามาเกี่ยวข้อง : Universe predictable

ตัวอย่างปัญหาที่มีความแน่นอน

- ปัญหาแปดปริศนา
- ทุกครั้งที่เลื่อนตัวเลข รู้ได้แน่นอนว่าจะเกิดอะไรขึ้น
- รู้สถานะต่อไปเมื่อใช้ตัวกระทำการหนึ่งๆ
- วางแผนการเล่นได้แน่นอน

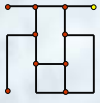
ตัวอย่างปัญหาที่มีความไม่แน่นอน

- การเล่นไพ่ มีการสุ่มเข้ามาเกี่ยวข้องในการหยิบไพ่
- ไม่รู้ว่าต่อไปจะได้ไบ่ไหน
- วางแผนการเล่นตายตัวไม่ได้
- วางแผนไว้หลายๆ ทาง แล้วเลือกทางที่น่าจะชนะที่สุด

รู้ว่าเป็นคำตอบได้โดยไม่ต้องเปรียบเทียบ

ตัวอย่างปัญหา (Absolute or relative)

- ปัญหาเขาวงกต
- ถ้าไม่มีเงื่อนไขไปสู่เป้าหมาย
- เส้นทางใดก็ตามที่ทำให้ไปสู่จุดหมาย ถือเป็นคำตอบของปัญหา
- ถ้ามีเงื่อนไขว่าต้องเป็นเส้นทางที่สั้นที่สุด
- ต้องนำคำตอบที่ได้เปรียบเทียบกับคำตอบอื่นๆ จนครบ จึงจะสามารถบอกได้ว่าเป็นคำตอบ
- ปัญหาแบบมีเงื่อนไขแก้ยาก และคำนวณมากกว่า



ฐานความรู้ที่ใช้มีความสอดคล้องกัน : Consistent

ตัวอย่างปัญหาที่ความรู้สอดคล้องกัน

- การแก้ปัญหาสัจพจน์ต่างๆ
  1. XY is defined for all XY
  2. X=Y, Y=Z -> X=Z
  3. X=X
  4. (XY)Z = X(YZ)
  5. IX=X
  6. $X^{-1}X=I$
  7. X=Y -> ZX = ZY
  8. X=Y -> XZ = YZ
- ต้องการพิสูจน์ XI=X
- ใช้วิธีการพิสูจน์ทางคณิตศาสตร์วิธีใดก็ได้

ตัวอย่างปัญหาที่ความรู้ไม่สอดคล้องกัน

- ปัญหาการคำนวณทางฟิสิกส์
  - สมชายยืนห่างเป้า 150 ฟุต ยิงเป้าด้วยความเร็วลูกปืน 1500 ฟุต/วินาที เขาต้องเล็งสูงกว่าเป้าเท่าใด
  - ความรู้ที่มีคือการคำนวณแบบเป็นเส้นตรง และความรู้เรื่องแรงดึงดูดโลก
  - ซึ่งความรู้ขัดแย้งกับความเป็นจริงที่ว่าลูกปืนวิ่งเป็นเส้นโค้ง
- ข้อเท็จจริงของฐานข้อมูลไม่สอดคล้องกัน เกิดการขัดแย้งกัน อาจทำให้แก้ปัญหาผิดพลาด

บทบาทของความรู้ : Role of knowledge

ตัวอย่างปัญหาที่ไม่ใช้ความรู้

- การเล่นหมากรุก
- คอมพิวเตอร์สามารถคำนวณได้เร็ว
- ไม่จำเป็นต้องใช้ความรู้ช่วย
- รู้กฎการเดิน และสร้างเส้นทางที่เป็นไปได้ทั้งหมด
- เลือกเดินตามเส้นทางที่ทำให้ชนะ
- ความรู้จะใช้เมื่อมีการจำกัดขอบเขตของการค้นหา หรือทำให้ได้คำตอบเร็วขึ้น

ตัวอย่างปัญหาที่ใช้ความรู้จำนวนมาก

- อ่านบทความแล้วสรุป
- อ่านบทความในหนังสือพิมพ์แล้วตัดสินว่าผู้เขียนสนับสนุนให้แก้ไขรัฐธรรมนูญหรือคัดค้าน
- ต้องใช้ความรู้จำนวนมาก เช่น
  - ข้อความในรัฐธรรมนูญ
  - สิ่งที่ผู้เขียนต้องการสอดคล้องกับข้อความในรัฐธรรมนูญหรือไม่

จำเป็นต้องใช้มนุษย์ช่วย : Human interaction

- บางปัญหา ต้องการให้คอมพิวเตอร์หาคำตอบให้โดยไม่สนใจขั้นตอนการหาคำตอบ
- บางปัญหา มนุษย์แก้ปัญหาได้ดีกว่ามาก
- อาจจะมีการโต้ตอบระหว่างมนุษย์กับคอมพิวเตอร์
- เพื่อช่วยแนะขั้นตอนการแก้ปัญหา ตัวอย่างเช่น
  - การพิสูจน์ทฤษฎีทางคณิตศาสตร์
  - การแปลจากภาษาหนึ่งเป็นอีกภาษาหนึ่ง

การค้นหาคำตอบ : Search

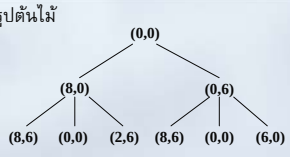
- สเปซของสถานะ คือ เซตของสถานที่เป็นไปได้ของระบบในระหว่างกระบวนการแก้ปัญหา
- การแก้ปัญหา คือ การใช้ตัวกระทำการเพื่อเปลี่ยนสถานะเริ่มต้นไปยังสถานะเป้าหมาย
- วิธีการแก้ปัญหา คือ ลำดับของตัวกระทำการที่นำมาใช้ต่อเนื่อง
- สถานะที่เป็นไปได้ทั้งหมดอาจมีจำนวนมาก เสียเวลาในการค้นหา
- การค้นหาอย่างมีประสิทธิภาพเป็นสิ่งสำคัญ
- ประสิทธิภาพของวิธีการค้นหาขึ้นกับลักษณะของปัญหาและความรู้ที่ใช้
- ข้อเสียของการค้นหาคำตอบคืออาจหาคำตอบไม่สำเร็จ ถ้าสเปซสถานะของปัญหามีขนาดใหญ่มาก

การแทนสเปซสถานะ : State space search

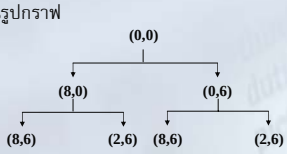
- เมื่อนิยามปัญหาในรูปสเปซสถานะแล้ว อาจแทนสเปซสถานะนั้นในรูป
  - ต้นไม้ (Tree)
  - กราฟ (Graph)
- สถานะต่างๆ จะแทนด้วยโหนด
- เมื่อใช้ตัวกระทำการ ทำให้สถานะเปลี่ยนไปจะแทนด้วยโหนดซึ่งมีเส้นโยงกับสถานะเก่า
- ตัวกระทำการใดเมื่อใช้แล้วทำให้กลับสู่สถานะเดิมก่อนหน้าสถานะปัจจุบันจะไม่รวมไว้ในกราฟ เพราะจะทำให้เส้นทางมีความยาวเป็นอนันต์ เพราะมีวัฏจักร (cycle) อยู่ด้วย
- การค้นหาคำตอบ คือ การสำรวจโหนดต่างๆ ในต้นไม้หรือกราฟนั่นเอง

ตัวอย่างการแทนสเปซสถานะ

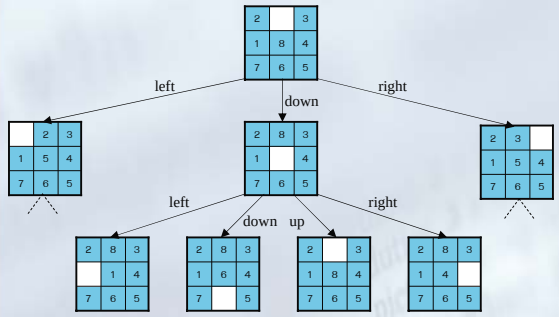
- ปัญหาถังน้ำในรูปต้นไม้



- ปัญหาถังน้ำในรูปกราฟ



ตัวอย่างการแทนสเปซสถานะ



วิธีการค้นหาคำตอบ

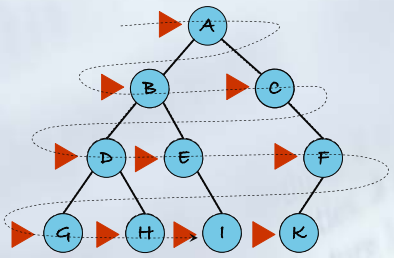
วิธีการค้นหาอาจแบ่งออกตามลักษณะได้ 4 ประเภท คือ

- **การค้นหาแบบพื้นฐาน** (Basic methods) เป็นการค้นหาแบบง่าย ใช้อัลกอริทึมทางโครงสร้างข้อมูลมาช่วย
- **การค้นหาแบบฮิวริสติก** (Heuristic methods) เหมาะกับการค้นหาที่มีสเปซสถานะขนาดใหญ่ ใช้ความรู้เข้ามาช่วยเลือกเส้นทาง
- **การค้นหาที่มีการกำหนดขอบเขตของการค้นหา** (Range constriction) ไม่กล่าวถึงในที่นี้
- **การค้นหาในการเล่นเกมต่าง ๆ** (Game playing) เป็นการค้นหาที่ต้องคำนึงถึงคู่ต่อสู้ด้วย

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)**
  - **การค้นหาดีที่สุดก่อน (Best-first search)**
  - **การค้นหาเอ-สตาร์ (A* algorithm)**
  - การค้นหาแบบอื่นๆ

การค้นหาแนวกว้างก่อน : Breadth-first search



การค้นหาแนวกว้างก่อน : Breadth-first search

- **Algorithm BFS**:

1. Create a variable NODE-LIST and set it to the 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 was empty, quit.
   - 2.2 For each way that each rule can match the state described in E do:
     - 2.2.1 Apply the rule to generate a new state.
     - 2.2.2 If the new state is a goal state, quit and return this state.
       Otherwise, add the new state to the end of NODE-LIST.

ข้อดีและข้อเสียของ BFS

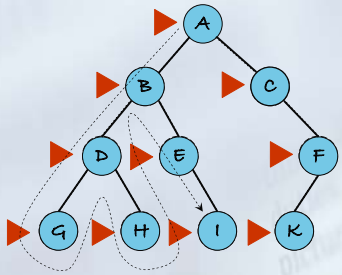
ข้อดี

- ถ้ามีคำตอบ ประกันได้ว่าจะพบคำตอบแน่
- คำตอบที่พบจะเป็นเส้นทางที่สั้นที่สุด
- ไม่ติดในเส้นทางที่ลึกมากๆ โดยไม่พบคำตอบ

ข้อเสีย

- ใช้หน่วยความจำมาก เพราะจำนวนโหนดในแต่ละระดับจะเพิ่มแบบ exponential
- เสียเวลามากในกรณีที่คำตอบอยู่ในระดับลึก
- ถ้าต้นไม้มีสถานะซ้ำๆ กันมาก จะทำให้ต้องสำรวจมากตามไปด้วย

การค้นหาแนวลึกก่อน : Depth-first search



การค้นหาแนวลึกก่อน : Depth-first search

- **Algorithm DFS:**

1. If the initial state is a goal state, quit and return success.

   Otherwise, 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, signal failure.

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

   1.3 If success is returned, signal success.

   Otherwise 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 problem

- ตัวกระทำการ คือ

1. กล่อง X ไม่มีกล่องอื่นทับ -> วาง X บนโต๊ะ

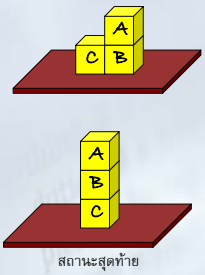
   CLEAR(X) -> ON(X,TABLE)

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

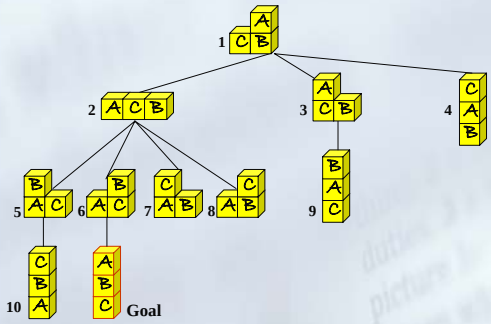
   CLEAR(X) & CLEAR(Y) -> ON(X,Y)

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

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



Block world : BFS



Block world : DFS