FACULTY OF ENGINEERING
CHULALONGKORN UNIVERSITY
2110327 ALGORITHM DESIGN
Year II, Second Semester, Final Examination, May 14, 2019 13:00-16:00

ชื่อ-นามสกุล.....เลขประจำตัว.....ตอนเรียนที่.....เลขที่ใน CR58.....

<u>หมายเหตุ</u>

1. ข้อสอบมีทั้งหมด 10 ข้อ ในกระดาษคำถาม 8 แผ่น 8 หน้า
2. **ไม่อนุญาตให้นำตำราและเอกสารใดๆ เข้าในห้องสอบ**
3. **ไม่อนุญาตให้ใช้เครื่องคำนวณใดๆ**
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่เจ้าหน้าที่ควบคุมการสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบและสมุดคำตอบออกจากห้องสอบ
6. ผู้เข้าสอบสามารถออกจากห้องสอบได้ หลังจากผ่านการสอบไปแล้ว 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. **นิสิตกระทำผิดเกี่ยวกับการสอบ ตามข้อบังคับจุฬาลงกรณ์มหาวิทยาลัย มีโทษ คือ พ้นสภาพการเป็นนิสิต หรือ ได้รับ สัญลักษณ์ F ในรายวิชาที่กระทำผิด และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้**

ห้ามนิสิตพกโทรศัพท์และอุปกรณ์สื่อสารไว้กับตัวระหว่างสอบ หากตรวจพบจะถือว่านิสิตกระทำผิดเกี่ยวกับการสอบ อาจต้องพ้นสภาพการเป็นนิสิต หรือ ให้ได้รับ F และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้

*** ร่วมรณรงค์การไม่กระทำผิดและไม่ทุจริตการสอบที่คณะวิศวกรรมศาสตร์ ***

ข้าพเจ้ายอมรับในข้อกำหนดที่กล่าวมานี้ ข้าพเจ้าเป็นผู้ทำข้อสอบนี้ด้วยตนเองโดยมิได้รับการช่วยเหลือหรือให้ความช่วยเหลือ ในการทำข้อสอบนี้

ลงชื่อนิสิต.....

วันที่.....

1. (25 คะแนน) จงระบุหน้าข้อความข้างล่างนี้ว่าเป็นข้อความที่ "จริง" หรือ "เท็จ" หรือเว้นว่างไว้ไม่ตอบ
ข้อที่ตอบถูก ได้ 1 คะแนน ตอบผิด ติดลบ 0.5 คะแนน (หากคะแนนรวมทุกข้อย่อยในข้อนี้เป็นลบ ให้ถือว่าข้อนี้ได้ 0 คะแนน)
กำหนดให้ กราฟ G ทั้งหลายในคำถามของข้อนี้แทนด้วย adjacency list โดย v และ e ของ G แทนจำนวนปมและจำนวนเส้นเชื่อม

_จริง_ สำหรับกราฟ G ใด ๆ DFS(G) ใช้เวลา $O(v+e)$ แต่ถ้า G เป็น connected undirected graph, DFS(G) ใช้เวลา $O(e)$

_จริง_ การใช้ DFS(G) เพื่อตรวจสอบว่า undirected graph G มี cycle หรือไม่ ใช้เวลา $O(v)$

_เท็จ_ directed graph G ที่มีเพียงปมเดียวที่มี in-degree เป็น 0 จะมี topological sort เพียงแบบเดียว

_เท็จ_ เมื่อเราทำ DFS(G, u) (คือทำ DFS กับกราฟ G เริ่มที่ u) แล้วสามารถเข้าถึงทุกปมในกราฟ G ได้หมด แสดงว่า G มี strongly connected component เพียง component เดียว

_จริง_ ถ้าเส้นเชื่อมใน G มีความยาวเท่ากันหมด เราสามารถหา single source shortest paths ของ G จากปม s ได้ในเวลา $O(v+e)$

_จริง_ ถ้าเรายุบรวมทุกปมที่อยู่ใน strongly connected component (SCC) เดียวกันให้เป็นเหลือปมเดียว (เช่นกราฟมี 3 SCC's หลังยุบแล้วจะเหลือ 3 ปม) กราฟหลังยุบแล้วต้องเป็น directed acyclic graph เสมอ

_จริง_ ต้นไม้ที่มี v ปมและเส้นเชื่อมทุกเส้นมีทิศทางจากปมพ่อ/แม่ไปยังปมลูก จะมี strongly connected component เป็นจำนวน v

_เท็จ_ เราใช้ priority queue ใน Prim algorithm เพื่อเก็บเส้นเชื่อมและความยาวของเส้นเชื่อม

_เท็จ_ เป็นที่ทราบกันว่า Dijkstra algorithm อาจให้คำตอบผิดถ้าเส้นเชื่อมบางเส้นมีความยาวติดลบ แต่ถ้าเส้นเชื่อม**ทุกเส้น**มีความยาวติดลบ Dijkstra algorithm จะให้คำตอบที่ถูกต้องเสมอ

_จริง_ หากนำกราฟ G ไปให้ Floyd-Warshall algorithm ทำงาน ได้ผลกลับมาเป็นเมทริกซ์ แล้วพบว่า มีสักช่องในแนวทแยงมุม (จากมุมซ้ายบนถึงขวาล่าง) เป็นจำนวนลบ สามารถสรุปได้ว่า G มี negative weight cycle

_เท็จ_ หากนำกราฟ G ที่มี negative weight cycle ไปให้ Bellman-Ford algorithm อาจทำให้เกิดการทำงานเป็นวงวนไม่รู้จบ

_เท็จ_ เราสามารถหา single source longest paths ในกราฟ G ได้ไม่ยาก ด้วยการสร้างกราฟใหม่ที่เหมือน G แต่เปลี่ยนความยาวของเส้นเชื่อมให้เป็นค่าติดลบของเส้นเชื่อมใน G แล้วใช้ Bellman-Ford algorithm หา single source shortest paths

_จริง_ การย้อนรอย (backtracking) เป็นการลดจำนวนสถานะที่ต้องพิจารณาใน state space โดยไม่ได้ช่วยอะไรเลยในการเลือกปมที่เหมาะสมให้นำไปสู่คำตอบได้รวดเร็ว

_จริง_ กำหนดให้ Q เป็นปัญหาหนึ่งที่หาคำตอบได้ด้วยการค้นคำตอบตามแนวลึก (depth-first search) ถ้าเรานำ least-cost search มาค้นคำตอบให้ปัญหา Q โดยให้ cost ของปมสถานะต่าง ๆ เป็นเลขสุ่ม ก็ยังคงหาคำตอบให้ปัญหา Q ได้

_เท็จ_ การกำหนดรูปแบบคำตอบที่ทำให้ได้ state space ที่ใหญ่ ย่อมทำให้การค้นคำตอบทำได้ช้ากว่าเสมอ เมื่อเทียบกับกรณีที่กำหนดรูปแบบคำตอบที่ได้ state space ที่เล็กกว่า ถึงแม้จะใช้กลวิธี backtracking ก็ไม่ช่วยมาก

_จริง_ ปัญหาตัดสินใจที่หาคำตอบได้รวดเร็ว ก็ย่อมตรวจคำตอบได้รวดเร็วด้วย

_จริง_ ปัญหาที่ถามว่า undirected graph G เป็น connected graph หรือไม่ จัดเป็นปัญหาในกลุ่ม NP

_เท็จ_ ปัญหาในกลุ่ม NP คือปัญหาตัดสินใจที่หาคำตอบได้ด้วย non-polynomial time algorithm

_จริง_ ปัญหา 2-colorable ถามว่า สามารถทาสีให้ปมต่าง ๆ ในกราฟด้วยสี 2 สี ได้หรือไม่ โดยไม่มีปมที่ปลายของเส้นเชื่อมเดียวกันมีสีเหมือนกัน เราสามารถจัดให้ปัญหา 2-colorable อยู่ในกลุ่มปัญหา P

_จริง_ ให้ Q และ R เป็นสองปัญหาที่ต่างกันในกลุ่ม P จะสรุปได้ว่า $Q \leq_P R$ และ $R \leq_P Q$

_จริง_ อัลกอริทึมที่แก้ปัญหาหนึ่งในกลุ่ม NP-Complete สามารถถูกนำไปปรับเพื่อแก้ปัญหาได้ทุกปัญหาในกลุ่มนี้

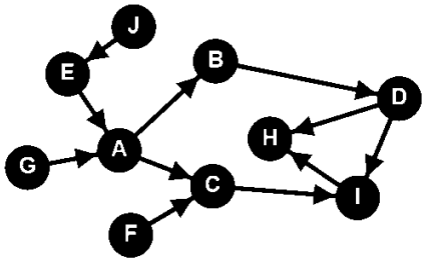
_เท็จ_ จากทฤษฎีของ Cook สรุปได้ว่าปัญหา satisfiability ลดรูป (ในเวลา polynomial time) ไปสู่ทุกปัญหาในกลุ่ม NP ได้หมด

_จริง_ ถ้าปัญหา Q1 ลดรูปไปเป็นปัญหา Q2 ได้ใน polynomial time ($Q1 \leq_P Q2$) แสดงว่าปัญหา Q1 ไม่ยากกว่าปัญหา Q2

_เท็จ_ เป็นที่รู้กันว่า $1SAT \in P$, $2SAT \in P$ แต่ $3SAT \notin P$

_จริง_ ถ้าปัญหา $Q1 \in NP$ และ Q2 เป็นปัญหา NP-hard โดย Q2 ลดรูปไปเป็นปัญหา Q1 ได้ใน polynomial time ($Q2 \leq_P Q1$) สรุปได้ว่า Q1 เป็นปัญหา NP-Complete

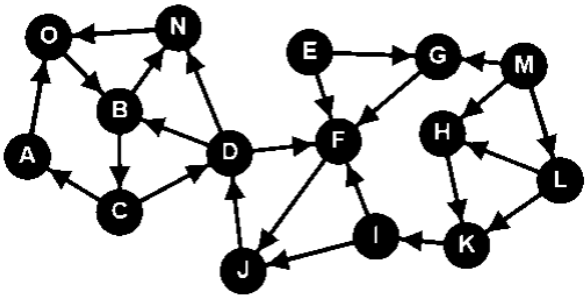
2. (5 คะแนน) จงเขียนลำดับของปมตาม topological sort ของกราฟทางขวานี้
(หากตอบได้หลายแบบ ให้เขียนมาหนึ่งแบบ)



Topological sort คือ _____

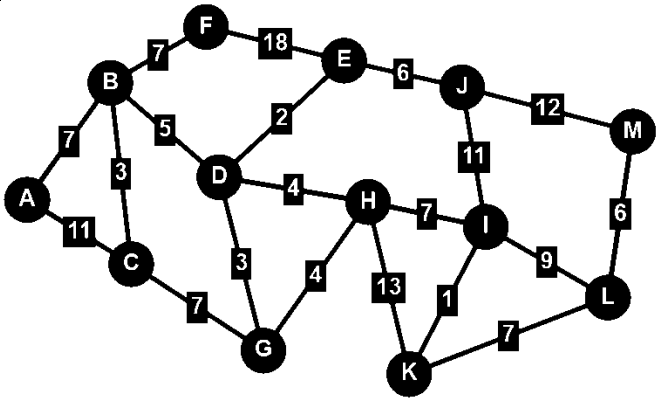
3. (5 คะแนน) จงเขียนปมของ strongly connected components ทั้งหมดของกราฟทางขวานี้
(หากตอบได้หลายแบบ ให้เขียนมาหนึ่งแบบ)

คำว่ากลุ่มในตารางข้างล่างนี้ คือ strongly connected component



	ชื่อปมในกลุ่ม		ชื่อปมในกลุ่ม		ชื่อปมในกลุ่ม
กลุ่มที่ 1		กลุ่มที่ 4 (ถ้ามี)		กลุ่มที่ 7 (ถ้ามี)	
กลุ่มที่ 2 (ถ้ามี)		กลุ่มที่ 5 (ถ้ามี)		กลุ่มที่ 8 (ถ้ามี)	
กลุ่มที่ 3 (ถ้ามี)		กลุ่มที่ 6 (ถ้ามี)		กลุ่มที่ 9 (ถ้ามี)	

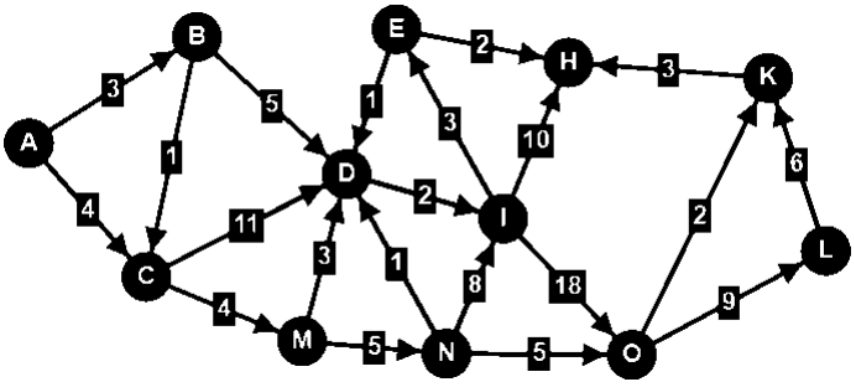
4. (5 คะแนน) จงเขียนเส้นเชื่อมตามลำดับที่ Kruskal algorithm นำเข้ามาอยู่ใน minimum spanning tree ของกราฟทางขวานี้
ให้เขียนเส้นเชื่อม โดยเขียนเฉพาะชื่อปมปลาย 2 ปมของเส้น เช่น เส้นระหว่างปม B กับปม D ก็ให้เขียนว่า BD
(หากตอบได้หลายแบบ ให้เขียนมาหนึ่งแบบ)



ลำดับของเส้นที่ถูกเพิ่มเข้าไปใน minimum spanning tree:

_____

5. (5 คะแนน) จงเขียนลำดับปมที่ Dijkstra algorithm นำเข้ามาอยู่ใน shortest path tree ของกราฟทางขวานี้ โดยเริ่มจากปม A
(หากตอบได้หลายแบบ ให้เขียนมาหนึ่งแบบ)



ลำดับของปมที่ถูกเพิ่มเข้าไปใน shortest path tree:

A,

_____

6. (10 คะแนน) สมมุติว่าเราได้คำนวณหา all pair shortest paths ของกราฟที่มี `n` ปมเรียบร้อยแล้ว โดยระยะทางที่สั้นสุดของวิถีระหว่างทุกคู่ของปมถูกเก็บในเมทริกซ์ `D[1..n][1..n]` จากนั้นได้มีการเติมปมที่ `n+1` เข้าไปในกราฟ โดยความยาวของเส้นระหว่างทุกคู่ของปมเก็บในเมทริกซ์ `W[1..(n+1)][1..(n+1)]` จงเขียนรหัสเทียมเพื่อคำนวณหาระยะทางที่สั้นสุดของวิถีระหว่างทุกคู่ของปม `n+1` ปม `E[1..(n+1)][1..(n+1)]` ที่ใช้เวลา $O(n^2)$ โดยรับประกันว่า

- `W[i][j]` $\geq$ `0` สำหรับทุก `i` กับ `j` คือไม่มี edge ที่น้ำหนักติดลบ
- หากไม่มี edge จาก ปม `i` ไปปม `j` แล้ว `W[i][j]` จะเป็น ∞
- `W[i][i]` เป็น `0` เสมอ

```
UpdateAllPairShortestPath( D[1..n][1..n], W[1..(n+1)][1..(n+1)] ) {
   E = new float array [1..(n+1)][1..(n+1)]









   return E
}
```

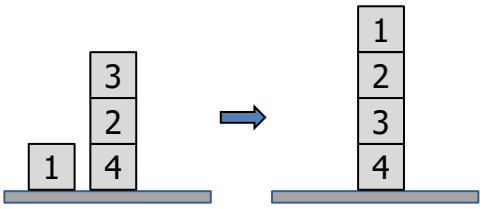
คำอธิบาย (หากมั่นใจว่า Code ทำงานถูกต้อง ไม่ต้องเขียนก็ได้)

_____

_____

_____

7. Blocks world เป็นเกมที่ประกอบด้วยกล่องหลายกล่อง แต่ละกล่องมีตัวเลขกำกับ กล่องถูกวางซ้อน ๆ กัน หลาย ๆ กองบนโต๊ะ ดังตัวอย่างที่แสดงในรูปข้างล่างนี้ โดยผู้เล่นต้องหาวิธีย้ายกล่อง เพื่อให้กล่องทุกกล่องวางซ้อนกันเป็นกองเดียวเรียงตามลำดับตัวเลขดังรูปทางขวา มีกฎการย้ายว่า ให้ย้ายกล่องได้ทีละใบ ห้ามมีอะไรอยู่บนกล่องที่ถูกย้าย กล่องจะถูกย้ายไปอยู่บนโต๊ะ หรือย้ายไปทับอีกกล่อง (ที่ต้องไม่มีอะไรอยู่บนกล่องนั้น) ก็ได้ เช่น จากสถานะ A ในรูป เราสามารถย้ายได้ 3 แบบ คือ ย้าย 1 ไปทับ 3, ย้าย 3 ไปทับ 1 และ 3 มาวางบนโต๊ะ



สถานะ A [0, 4, 2, 0] สถานะสุดท้ายที่ต้องการ [2, 3, 4, 0]

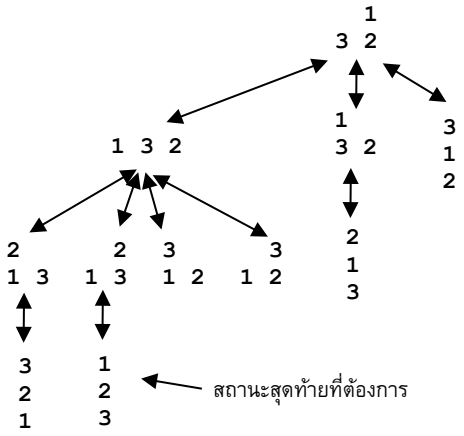
ถ้ามีกล่อง *n* ใบ เราสามารถแทนสถานะการวางกล่องด้วยอาเรย์ชื่อ **under** ขนาด *n* ช่อง (ช่องที่ 1 ถึง *n*) โดย **under[k]** เก็บหมายเลขของกล่องที่ถูกด้านล่างกล่อง **k** ถ้า **under[k]** เป็น 0 แสดงว่า กล่อง **k** วางบนโต๊ะ เช่น สถานะ A ในรูป มี **under = [0,4,2,0]** แทนว่า ใต้กล่อง 1 คือโต๊ะ, ใต้กล่อง 2 คือกล่อง 4, ใต้กล่อง 3 คือกล่อง 2, และ ใต้กล่อง 4 คือโต๊ะ

ฟังก์ชันที่น่าจะใช้ไปเป็นประโยชน์ เขียนให้แล้ว ก็เรียกใช้ได้เลย

```
// สถานะ under นี้เป็นสถานะสุดท้าย ?
is_goal_state( under[1..n] ) {
  for (i = 1 to n-1) {
     if (under[i] != i+1)
       return false
  }
  return under[n] == 0
}

// กล่อง e อยู่บนโต๊ะ ?
on_desk( under[1..n], e ) {
  return under[e] == 0
}

// ขอกล่องทั้งหมดที่อยู่บนสุดของทุก ๆ กอง
top_blocks( under[1..n] ) {
  all = set( 1,2,3,...,n )
  not_top_blocks = set( under )
  tops = all - not_top_blocks
  return tops
}
```

ก) (5 คะแนน) จงเขียนฟังก์ชัน **next_states** ที่ผลิตลิสต์ของสถานะถัดไปทั้งหมดที่ผลิตได้จากสถานะที่ได้รับ (ซึ่งคือค่าของ **under**) ตัวอย่างเช่น จาก state space ที่แสดงข้างล่างนี้ **under** ที่รากคือ [2,0,0] จะผลิตสถานะถัดไปคือ [[0,0,0], [3,0,0] และ [2,0,1]]

```
next_states( under[1..n] ) {     // สามารถเรียกใช้ฟังก์ชันข้างบนนี้ ให้เป็นประโยชน์ได้
    states = an empty list (or vector)

    tops = top_blocks(under)
    for (a : tops) {
        if (under[a] != 0) {
            new_under = clone(under)
            new_under[a] = 0
            states.push_back( new_under )
        }
        for (b : tops) {
            if a != b:
                new_under = clone(under)
                new_under[a] = b
                states.push_back( new_under )
        }
    }

    return states
}
```



ข) (5 คะแนน) จงเขียนฟังก์ชัน **solve** ที่รับสถานะเริ่มต้น แล้วค้นให้พบสถานะสุดท้าย (เขียนแค่ค้นจนพบก็พอ ไม่ต้องแสดงวิธีการย้ายกล่อง พอเจอก็แสดงคำว่า OK เท่านี้พอ) โดยเขียนให้ค้นแบบ breadth-first search (ต้องหลีกเลี่ยงการค้นสถานะซ้ำ ๆ)

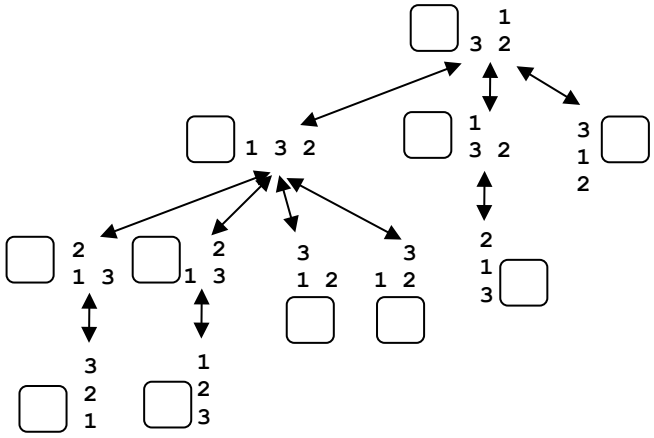
```
solve( under[1..n] ) {     // under ที่รับมานี้เป็นสถานะเริ่มต้น

      states = new set()
      q = new queue()
      q.push(under)
      states.insert(under)
      while len(queue) > 0:
          u = queue.pop()
          if is_goal_state(u):
              print("OK")
              return
          for s in next_states(u):
              if s not in states:
                  q.push(s)
                  states.insert(s)
      print("Not Found")


}
```

ค) (5 คะแนน) ถ้าต้องการค้นคำตอบของ Blocks world ด้วย least cost search จงเสนอ cost function ที่สมเหตุสมผลในการนำทาง (คือเลือกปมที่เหมาะสม) เพื่อค้นพบคำตอบได้ดี ให้อธิบายวิธีการคำนวณ cost function พร้อมตัวอย่าง เช่น สถานะ [2, 4, 0, 5, 6, 0] กับ [0, 1, 2, 3, 4] และให้เขียน cost ตามวิธีที่เสนอในกล่องสี่เหลี่ยมกำกับปมต่าง ๆ ของ state space ในรูปข้างล่างนี้ด้วย

ให้ cost ของปมเท่ากับจำนวนกล่อง k ที่มี under[k] ไม่เท่ากับ (k+1)%(n+1)

ตัวอย่าง: สถานะ [2, 4, 0, 5, 6, 0] มี cost = 2,

สถานะ [0, 1, 2, 3, 4] มี cost = 5



8. ปัญหา **M**issionaries and **C**annibals: กำหนดให้เริ่มต้นมีบาทหลวง (ขอเขียนสั้น ๆ ว่า M) 3 คน มนุษย์กินคน (ขอเขียนว่า C) 3 คน และเรือ 1 ลำ อยู่ทางฝั่งซ้ายของแม่น้ำ ทั้ง 6 คนต้องการนั่งเรือข้ามไปยังฝั่งขวาของแม่น้ำ แต่เรือลำนี้นั่งได้อย่างมาก 2 คน ถ้า ณ ขณะใดขณะหนึ่งที่ฝั่งซ้ายหรือฝั่งขวาของแม่น้ำ มี C เป็นจำนวนมากกว่า M C ก็จะกิน M ที่ฝั่งนั้น สิ่งที่ต้องการคือ ลำดับการนั่งเรือข้ามฝั่งไปมา เพื่อให้คนทั้ง 6 ข้ามมายังฝั่งขวาได้สำเร็จทุกคน (ไม่ตายด้วย) ขอกำหนดให้รูปแบบของสถานะเป็นดังข้างล่างนี้ (เช่น (3, 2, L, 0, 1))

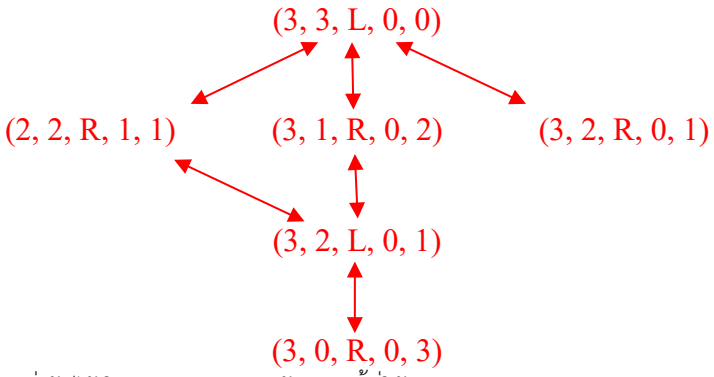
(*จำนวน M ที่ฝั่งซ้าย, จำนวน C ที่ฝั่งซ้าย, ตำแหน่งของเรือ L (ซ้าย) หรือ R (ขวา) ของแม่น้ำ, จำนวน M ที่ฝั่งขวา, จำนวน C ที่ฝั่งขวา*)

ขอนิยามให้ dead state คือ สถานะที่มีจำนวน C มากกว่าจำนวน M (ซึ่งอาจเกิดทางฝั่งซ้าย หรือฝั่งขวาก็ได้) เช่น จากสถานะ (3, 2, L, 0, 1) ซึ่งไม่ใช่ dead state ถ้า M 1 คนกับ C 1 คนขึ้นเรือไปฝั่งขวา พอถึงฝั่งก็เป็น (2, 1, R, 1, 2) ซึ่งเป็น dead state

ก) (1 คะแนน) state เริ่มต้นใน state space คือ (3, 3, L, 0, 0)

ข) (1 คะแนน) state ที่ต้องการค้นให้พบ คือ (0, 0, R, 3, 3)

ค) (5 คะแนน) จงวาด state space สัก 4 ระดับ (ให้ state เริ่มต้นอยู่ในระดับที่ 1) โดยไม่ต้องแสดง dead states ไม่แสดง state ซ้ำ แต่ให้เขียนเส้นที่มีลูกศรเพื่อแสดงให้เห็นทิศทางว่า state ใด ผลิตได้จาก state ใด (แบบเดียวกับรูป state space ในหน้าที่ 5)



9. ก) (4 คะแนน) จงเติมคำสั่งในช่องว่างที่เว้นไว้ ใน pseudo code ข้างล่างนี้ที่ใช้หาคำตอบของปัญหา sum of subset (แบบ decision problem) ด้วยวิธีการค้นแบบ depth-first search

```
sum_of_subset( d[1..n], k ) {  คืน True เมื่อมีเซตย่อยของ d ที่ผลรวมเท่ากับ k ถ้าไม่มีคืน False
  return sss_R( d, k, new int[1..n], 0 )
}
sss_R( d[1..n], k, x[1..n], m ) {
  if (m == n) {
     if ∑_{j=1}^{m} x[j]d[j] == k: return True
  } else {

     x[m+1] = 0
     if (sss_R(d, k, x, m) == True) return True
     x[m+1] = 1
     return sss_R(d, k, x, m)

  }
}
```

ข) (4 คะแนน) ในการแก้ปัญหา 0/1 Knapsack ด้วยวิธี branch and bound ที่ได้เรียนในวิชานี้ หากปัญหาที่ได้รับมีน้ำหนักของแต่ละชิ้นเป็น w = [30, 10, 20, 50, 40], มูลค่าของแต่ละชิ้นเป็น v = [66, 20, 30, 60, 40] และถุงรับน้ำหนักได้มากสุด 100 อยากทราบว่า upper bound ของมูลค่ารวมที่สถานะ [0, 1, -, -, -] และ สถานะ [1, 0, -, -, -] มีค่าเท่าไร (คำนวณด้วยวิธีที่นำเสนอในห้องเรียน) (หมายเหตุ: 66/30 = 2.2, 20/10 = 2.0, 30/20 = 1.5, 60/50 = 1.2, 40/40 = 1.0)

- upper bound ของมูลค่ารวมที่สถานะ [0, 1, -, -, -] 20 + (30 + 60 + 0.5x40) = 130
- upper bound ของมูลค่ารวมที่สถานะ [1, 0, -, -, -] 66 + (30 + 60) = 156

10. ก) (5 คะแนน) ให้ปัญหา Q รับ weighted directed graph G กับค่า K เพื่อหาว่า มี path ใน G หรือไม่ ที่มีความยาวไม่เกิน K จงแสดงให้เห็นจริงว่า ปัญหา Q ∈ NP

```
Q( G[1..n][1..n], K, P[1..m] ) {
  s = 0
  for (i = 1 to m-1) {
    s += G[ P[i] ][ P[i+1] ]
  }
  return s <= K
}
```

ข) (5 คะแนน) สมมติว่า DSSS(d[1..n], k) เป็นฟังก์ชันแบบ polynomial time ที่คืนค่าจริง ถ้า d มีเซตย่อยที่มีค่ารวมเป็น k ถ้าไม่มี ก็คืนค่าเท็จ จงนำ DSSS ไปใช้เพื่อออกแบบ polynomial time algorithm (เขียนเป็น pseudo code) เพื่อหาเซตย่อยของ d ที่มีค่ารวมเป็น k (ในกรณีที่ไม่มีเซตย่อยที่ต้องการ ให้คืน NULL)

```
SSS( d[1..n], k ) {
  if (DSSS(d, k) == False) return NULL
  soln = []; dd = copy of d
  for (e : d) {
     dd.remove(e)
     if (DSSS(dx, k) == False) {
        SS.push(e); k -= e
     }
  }
  return SS
}
```

ค) (5 คะแนน) ข้อนี้เกี่ยวกับปัญหา TSP กับ HAM TSP รับ complete undirected weighted graph G และค่า K เพื่อตอบว่า มีวงจรในกราฟ G ที่ผ่านทุกปม ปมละครั้ง และมีความยาวรวมไม่เกิน K หรือไม่ ส่วน HAM รับ undirected graph H เพื่อตอบว่า มีวงจรในกราฟ H ที่ผ่านทุกปม ปมละครั้งหรือไม่ จงพิสูจน์ว่า TSP is NP-complete ถ้ารู้แล้วว่า TSP ∈ NP และ HAM is NP-complete

เนื่อง HAM เป็น NP-complete และ TSP อยู่ใน NP จะแสดงแค่ว่า HAM $\leq_P$ TSP

เราสามารถแปลงกราฟ H ของ HAM ให้เป็น G และ K ของ TSP ดังนี้

G สร้างให้เป็น complete graph มีจำนวนปมเท่ากับของ H มีน้ำหนักกำกับเส้นเชื่อม ดังนี้

$$w_G(i,j) = \begin{cases} 0 & \text{ถ้ามีเส้น } (i,j) \text{ ใน } H \\ 1 & \text{ถ้าไม่มีเส้น } (i,j) \text{ ใน } H \end{cases}$$

และให้ K เท่ากับ 0 จะได้ว่า HAM(H) is true iff TSP(G,0) is true เพราะ

ถ้า HAM(H) ตอบ True วงจรฮามิลตันของ H ย่อมมีผลรวมน้ำหนักเป็น 0 ใน G ดังนั้น TSP(G,0) ตอบ True

ถ้า TSP(G,0) ตอบ True ก็ต้องมีวงจรที่เส้นเป็น 0 หมด ซึ่งต้องเป็นเส้นที่มีใน H ด้วย ดังนั้น HAM(H) ตอบ True