Greedy Algorithms

Slides by

กันต์ ศรีจันท์ทองศิริ

สถาบันเทคโนโลยีนานาชาติสิรินธร

Greedy Algorithm คืออะไร

- อัลกอริทึมสำหรับปัญหาประเภท optimization หลาย ๆ ข้อ มักจะประกอบด้วยหลายขั้นตอน
- แต่ละขั้นตอนจะต้องมีการตัดสินใจบางอย่าง
- **Greedy Algorithm** คือประเภทของอัลกอริทึมที่มีแนวคิดว่าในแต่ละขั้น จะตัดสินใจเลือกสิ่งที่**ดูเหมือนเป็นสิ่งที่ดีที่สุด**ในตอนนั้น ๆ เสมอ

- (Greedy = โลภ)

- สำหรับบางปัญหา, Greedy Algorithms จะให้คำตอบสุดท้ายที่ดีที่สุดจริง ๆ (Globally optimal solution).
- แต่หลาย ๆ ปัญหา การใช้ Greedy Algorithms จะไม่ได้คำตอบที่ดีที่สุด
  - แต่ก็อาจจะเป็นคำตอบที่ดีพอสำหรับบางกรณี
    - อย่างเช่นปัญหาเป็น NP-Hard แต่มีวิธี Greedy ง่าย ๆ ที่ให้คำตอบที่ใกล้เคียงกับ Optimal solution

- **ข้อดี:**
  - มักจะง่ายในการเขียนโปรแกรม
  - โปรแกรมมักจะมี Time Complexity ต่ำ
    - เพราะสิ่งที่เลือกไปแล้ว มักจะไม่กลับมาเปลี่ยนการตัดสินใจทีหลัง และ
    - การเลือกสิ่งที่ดีที่สุดในขณะหนึ่ง ๆ มักทำได้ง่าย
- **ข้อเสีย:**
  - มักไม่ได้ global optimal solution ในหลาย ๆ ปัญหา

- เพราะฉะนั้น เมื่อเจอปัญหาใหม่ ๆ ขั้นแรกควรลองดูว่าสามารถใช้หลักการ **Greedy** ในการแก้ปัญหาได้เลยหรือไม่ และได้คำตอบที่ **Optimal** หรือไม่ (อาจจะต้องพิสูจน์)
  - ถ้าได้ ก็ใช้เลย
  - ถ้าไม่ได้ ค่อยลองอัลกอริทึมประเภทที่ซับซ้อนกว่าแทน

ปัญหาแคชเชียร์ทอนเงิน

- แคชเชียร์มีเหรียญ 1 บาท, 2 บาท, 5 บาท, และ 10 บาท จำนวนไม่จำกัด (ไม่มีธนบัตร)
- ต้องการทอนเงิน **W** บาท ให้ใช้จำนวนเหรียญในการทอนน้อยที่สุดต้องทอนอย่างไร

วิธีที่คนทั่วไปใช้:

1. ทอนเหรียญ 10 บาทให้มากที่สุด โดยไม่เกินจำนวนเงินที่ต้องทอน
2. หลังจากนั้น ทอนจำนวนเงินที่เหลือด้วยเหรียญ 5 บาทให้มากที่สุด โดยไม่เกินจำนวนเงินที่ต้องทอน
3. ทำเหมือนเดิมกับเหรียญ 2 บาท และ 1 บาท...

ตัวอย่างการทำงานของวิธีนี้

- ถ้าต้องทอน W = 18 บาท
- ทอนเหรียญ 10 บาทหนึ่งเหรียญ (ถ้า 2 เหรียญจะเกิน 18 บาท)
- เหลือ 8 บาท ทอนเหรียญ 5 บาทหนึ่งเหรียญ (ถ้า 2 เหรียญจะเกิน 8 บาท)
- เหลือ 3 บาท ทอนเหรียญ 2 บาทหนึ่งเหรียญ
- เหลือ 1 บาท ทอนเหรียญบาท หนึ่งเหรียญ
- ทั้งหมดใช้ 4 เหรียญ

- วิธีนี้เป็นวิธี Greedy
  - การทอนโดยเหรียญ 10 บาทให้มากที่สุดโดยไม่เกินจำนวนที่ต้องทอน เป็นสิ่งที่ดูเหมือนดีที่สุดในขณะนั้น (โดยไม่ได้คำนึงถึงเงินที่เหลือที่ต้องทอนด้วยเหรียญอื่น ๆ)

Optimal หรือไม่?

- วิธีนี้ได้คำตอบ Optimal ถ้าเป็นเงินไทย (และเงินสกุลต่าง ๆ ส่วนมากในโลกนี้)
- ยัง Optimal ถึงแม้รวมธนบัตรทั้งหมดเข้าไปด้วย
- เมื่อไหร่ วิธีนี้ถึงจะไม่ Optimal?
  - สมมติว่ามีเหรียญ 4 บาทด้วย

- ถ้าต้องทอน 8 บาท (มีเหรียญ 5 บาท, 4 บาท, 2 บาท, 1 บาท)
- วิธี Greedy จะทอนอย่างไร? ใช้กี่เหรียญ?
- คำตอบ Optimal คือ?

- สาเหตุที่ไม่มีเหรียญ 4 บาท

จอดปั้มไหนดี

- บริษัทที่คุณทำงานอยู่ต้องการพาพนักงานไปเที่ยวโดนการขับรถไปหนึ่งคัน หัวหน้าต้องการเอาใจพนักงานจึงสัญญาว่าหากรถแวะที่ปั๊มไหน บริษัทจะจ่ายเงินค่าขนมที่พนักงานซื้อที่ปั๊ม แต่หัวหน้าไม่อยากจ่ายเงินเยอะมาก จึงไม่อยากจะหยุดทุกปั๊มแต่จะหยุดที่ปั๊มก็ต่อเมื่อจำเป็นต้องเติมน้ำมัน
- จงเขียนโปรแกรมเพื่อช่วยในการตัดสินใจว่าต้องหยุดปั๊มไหนบ้าง โดยมีข้อมูลเข้าเป็นแผนที่ ของเส้นทางและปั๊มระหว่างทาง ระยะห่างระหว่างปั๊ม และระยะทางที่รถจะวิ่งได้เมื่อเริ่มต้นจากน้ำมันเต็มถัง โดยจะต้องหยุดปั๊มเท่าที่จำเป็น

จับคู่นักสกี

- ณ.ลานสกีแห่งหนึ่งมีนักสกีจำนวนหนึ่งและมีอุปกรณ์อยู่จำนวนหนึ่ง ทั้งนี้การจะเล่นสกีได้ดี ขนาดความยาวของอุปกรณ์เล่นและความสูงของผู้เล่นจะต้องไม่แตกต่างกันมาก เจ้าของลานสกีจึงมาขอร้องให้คุณช่วยเขียนโปรแกรมเพื่อจับคู่นักสกีและอุปกรณ์สกี โดยต้องการให้ผลรวมของความแตกต่างระหว่างขนาดความยาวของอุปกรณ์เล่นและความสูงของผู้เล่นน้อยที่สุด
- ข้อมูลเข้า
  - ชื่อและความสูงของนักสกีแต่ละคน
  - หมายเลขอุปกรณ์สกีแต่ละชุดและความยาว
- ผล: รายชื่อนักสกีและหมายเลขอุปกรณ์ที่คู่กัน

Remark: Greedy Algorithms

- Selecting a locally optimum choice at each stage with the hope of finding the global optimum
- the question is when to use a greedy algorithm
- Greedy choice property
  - The choice make by an greedy algorithm may depend on choice so far, but it cannot depend on any future choices or on the solutions to sub-problems

Other greedy algorithms

- Dijkstra's algorithm for finding the shortest path in a graph
  - Always takes the *shortest* edge connecting a known node to an unknown node
- Kruskal's algorithm for finding a minimum-cost spanning tree
  - Always tries the *lowest-cost* remaining edge
- Prim's algorithm for finding a minimum-cost spanning tree
  - Always takes the *lowest-cost* edge between nodes in the spanning tree and nodes not yet in the spanning tree