แนวคิดเชิงคำนวณและการออกแบบอัลกอริทึม
Computational thinking and algorithm design

ผู้ช่วยศาสตราจารย์สมเกียรติ ช่อเหมือน (tko@webmail.npru.ac.th)

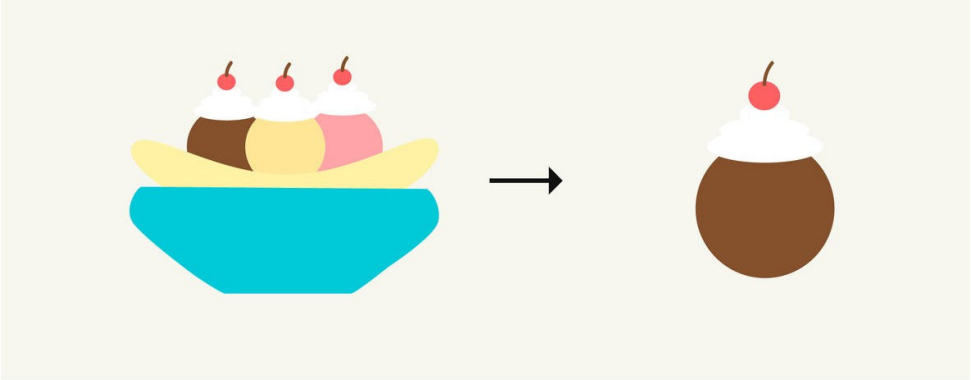
สาขาวิชาวิศวกรรมซอฟต์แวร์ คณะวิทยาศาสตร์และเทคโนโลยี

บทนำ

- แนวคิดเชิงคำนวณ
  - ความคิดรวบยอด
  - การแบ่งปัน
  - การแปลง
  - การเปรียบเทียบ
- การออกแบบอัลกอริทึม

imageURL

แนวคิดเชิงคำนวณ



ความคิดรวบยอด (Abstraction)

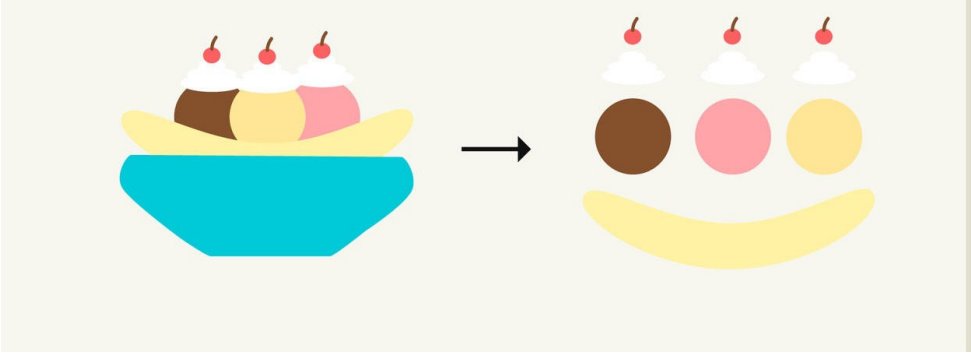
การรวบรวมข้อมูลหรือกิจกรรมที่สำคัญ เพื่อสร้างโมเดลหรือผลลัพธ์ที่มีประสิทธิภาพ

ในการพัฒนาโปรแกรม

ช่วยให้โปรแกรมเมอร์มุ่งเน้นไปที่ปัญหาหลักและละเว้นรายละเอียดที่ไม่จำเป็น

imageURL

แนวคิดเชิงคำนวณ



การแบ่งปัน (Decomposition)

การแบ่งการทำงานออกเป็นหน่วยย่อย ๆ ทำให้งานที่ซับซ้อนทำได้ง่ายขึ้น

การแบ่งโปรเจ็กต์ใหญ่ออกเป็นหน้าที่ย่อย ๆ ทำให้สามารถจัดการและทำงานร่วมกันได้

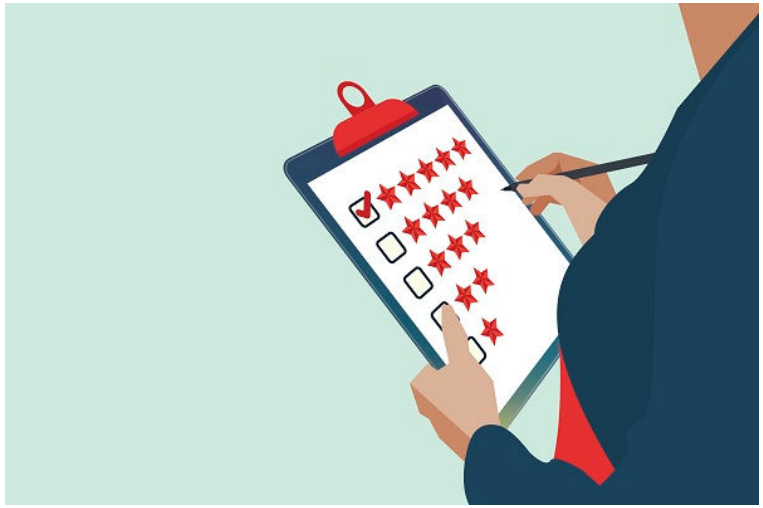
แนวคิดเชิงคำนวณ

การแปลง (Transformation)

การเปลี่ยนแปลงข้อมูลหรือแนวคิดเพื่อให้สอดคล้องกับเป้าหมายหรือประสบการณ์ใหม่

การแปลงข้อมูลจากรูปแบบหนึ่งไปยังอีกรูปแบบเพื่อให้โปรแกรมมีประสิทธิภาพ

แนวคิดเชิงคำนวณ

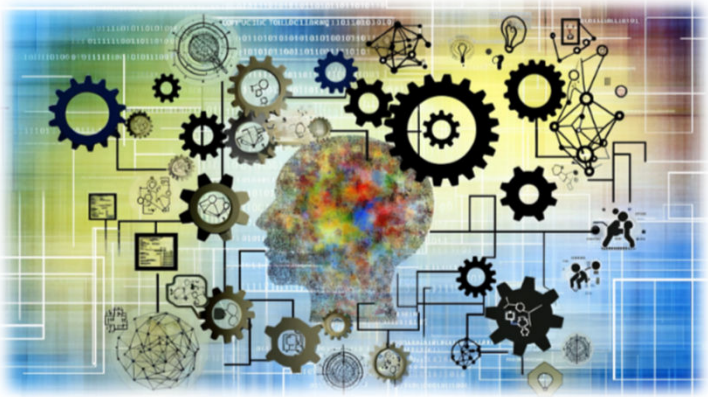


การเปรียบเทียบ (Evaluation)

การวิเคราะห์ข้อมูลหรือปัญหาในมุมมองต่าง ๆ เพื่อหาวิธีที่ดีที่สุดในการแก้ปัญหา

การวิเคราะห์ประสิทธิภาพของโปรแกรมหรือการวิจารณ์โค้ดเพื่อหาวิธีในการปรับปรุง

แนวคิดเชิงคำนวณ



- ปัญหาที่คอมพิวเตอร์สามารถแก้ไขได้
- การแก้ไขปัญหาที่มีประสิทธิภาพ
- ขอบเขตการคำนวณของคอมพิวเตอร์
- ประเภทของปัญหาที่คอมพิวเตอร์สามารถแก้ไขได้
- ความต้องการทรัพยากรทางคอมพิวเตอร์
- ปัญหาที่คอมพิวเตอร์ไม่สามารถหาคำตอบได้ในเวลาที่จำกัด

ปัญหาที่คอมพิวเตอร์สามารถแก้ไขได้

- ปัญหาที่ตัดสินได้ **(decidable problems)**
- ปัญหาที่เข้าถึงได้ด้วยวิธีการทางคอมพิวเตอร์ **(computationally solvable problems)**
- ปัญหาที่มีขั้นตอนวิธีหรืออัลกอริทึมที่ชัดเจน
  - การคำนวณทางคณิตศาสตร์
  - การเรียงลำดับข้อมูล
  - การค้นหา
  - การประมวลผลภาษาธรรมชาติ
  - การแก้ปัญหาการเดินทาง
  - การแก้ปัญหาการแบ่งงาน
  - เป็นต้น

ขอบเขตการคำนวณของคอมพิวเตอร์

- ทรัพยากรฮาร์ดแวร์
- ความซับซ้อนเวลา
- ทฤษฎีความซับซ้อนคอมพิวเตอร์
- ข้อจำกัดทางทฤษฎี
- ประสิทธิภาพอัลกอริทึม

ทรัพยากรทางคอมพิวเตอร์ในกระบวนการพัฒนา

- ฮาร์ดแวร์
- ซอฟต์แวร์
- เครื่องมือพัฒนา
- การเข้าถึงทรัพยากรออนไลน์
- เอกสารและทรัพยากรการเรียนรู้
- เวลาและทรัพยากรทางกายภาพ

การออกแบบอัลกอริทึม

- กระบวนการสร้างชุดของขั้นตอนวิธีหรือคำสั่งให้คอมพิวเตอร์
- การออกแบบอัลกอริทึม
  - หาวิธีการแก้ไขปัญหา
  - หาวิธีที่มีประสิทธิภาพที่สุด
  - ต้องพิจารณาถึงความชัดเจนของขั้นตอน
  - ความถูกต้องของผลลัพธ์
  - ประสิทธิภาพทั้งในด้านเวลาและหน่วยความจำ

การออกแบบอัลกอริทึม

1. ขั้นตอน (Steps)

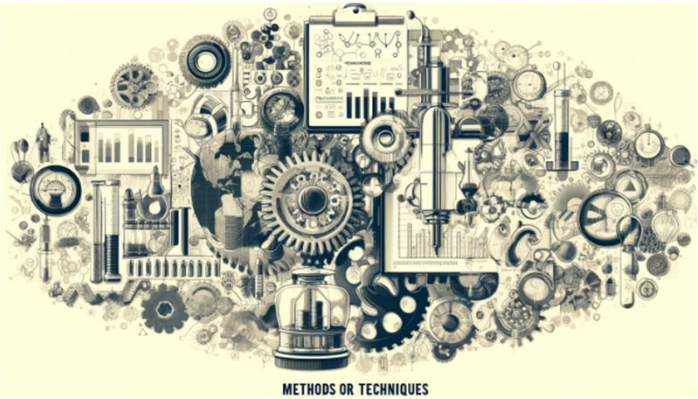
- ขั้นตอนหรือลำดับการดำเนินการที่ชัดเจนในการแก้ปัญหา
- ขั้นตอนสามารถประมวลผลได้และสามารถติดตามได้ง่าย

2. ปัญหา (Problem Identification and Analysis)

- ระบุปัญหาที่ต้องการแก้ไข การออกแบบอัลกอริทึมจากปัญหา
- วิเคราะห์ปัญหา เพื่อให้เข้าใจลักษณะ ขอบเขต และความซับซ้อน

3. วิธีการ (Methods or Techniques)

- เลือกวิธีหรือเทคนิคในการแก้ปัญหา
- เทคนิคหรือวิธีการนี้อาจจะรวมถึง
    - การใช้โครงสร้างข้อมูล
    - การคำนวณ
    - การประมวลผลทางตรรกศาสตร์.



กระบวนการสร้างชุดของขั้นตอนวิธีหรือคำสั่ง

- การวิเคราะห์ปัญหา (Problem Analysis)
- การกำหนดข้อกำหนด (Specification)
- การออกแบบอัลกอริทึม (Algorithm Design)
- การพิสูจน์อัลกอริทึม (Algorithm Verification)
- การประเมินประสิทธิภาพ (Performance Evaluation)
- การทดสอบและการดีบัก (Testing and Debugging)
- การปรับปรุงและการคิดค้นใหม่ (Refinement and Iteration)

1) การวิเคราะห์ปัญหา

- ปัญหา: หาจำนวนคู่ของชุดตัวเลขที่ผลรวมเท่ากับค่าที่กำหนด
- ข้อมูลนำเข้า (Inputs)
  - ชุดของตัวเลข (numbers)
  - ค่าเป้าหมาย targetSum
- ผลลัพธ์ที่ต้องการ (Outputs)
  - รายการของคู่ตัวเลขที่ผลรวมเท่ากับ targetSum

ขั้นตอนวิเคราะห์(1)

- เข้าใจปัญหา:
  - คำถามคืออะไร ต้องการหาคู่ตัวเลขที่มีผลรวมเท่ากับค่าใดค่าหนึ่ง
  - ข้อจำกัดหรือเงื่อนไขอะไรบ้าง
  - ค่าเป้าหมายมีลักษณะอย่างไร
  - ตัวเลขในชุดตัวเลขสามารถมีค่าซ้ำกันได้หรือไม่

ขั้นตอนวิเคราะห์ (2)

- กำหนดข้อจำกัดและเงื่อนไข
  - ต้องคืนค่าคู่ตัวเลขในรูปแบบใด
    - อาร์เรย์ของอาร์เรย์
    - รายการของคู่ตัวเลข
    - ชุดของตัวเลขที่มีคู่ของดัชนีที่ชี้ไปยังตำแหน่งของตัวเลขในอาร์เรย์
  - ต้องจัดการกับข้อมูลนำเข้าที่มีขนาดใหญ่อย่างไร
  - มีการจำกัดเวลาหรือทรัพยากรหน่วยความจำหรือไหม

ขั้นตอนวิเคราะห์(3)

- สร้างตัวอย่าง
  - ตัวอย่างชุดของตัวเลขและค่าเป้าหมาย
    - [1, 2, 3, 4, 5] กับ targetSum เท่ากับ 5 คำตอบที่คาดหวังคือ [[1, 4], [2, 3]]
- พิจารณาข้อผิดพลาดและกรณีพิเศษ
  - ถ้าชุดตัวเลขว่างเปล่า หรือไม่มีคู่ใดที่ผลรวมเท่ากับ targetSum เลย จะต้องจัดการอย่างไร
  - หากมีคู่ตัวเลขหลายคู่ที่ผลรวมตรงกับ targetSum จะแสดงผลลัพธ์อย่างไร
- โครงสร้างขั้นต้นของอัลกอริทึม
  - อัลกอริทึมการค้นหาใช้เวลา $O(n^2)$
  - ปรับปรุงรูปแบบที่มีประสิทธิภาพ
    - การใช้ HashMap เพื่อลดเวลาทำงานลง

2) ข้อกำหนดสำหรับปัญหา

- ประเภทของข้อมูลนำเข้า
  - ชุดตัวเลข อาร์เรย์ของจำนวนเต็ม numbers ซึ่งตัวเลขซ้ำได้
  - ค่าเป้าหมาย จำนวนเต็ม targetSum ที่คู่ตัวเลขที่ผลรวมเท่ากับค่านี้
- ข้อกำหนดเกี่ยวกับข้อมูลออก
  - ผลลัพธ์ คือ ชุดของคู่ตัวเลขที่ผลรวมเท่ากับ targetSum
  - ตัวเลขแต่ละคู่ในผลลัพธ์ควรเป็นอาร์เรย์ที่มีสองตัวเลขจาก numbers
- ข้อจำกัดเกี่ยวกับผลลัพธ์
  - คู่ตัวเลขสามารถใช้ตัวเลขซ้ำได้ หากมีตำแหน่งที่แตกต่างกันในอาร์เรย์
  - ผลลัพธ์ไม่ควรมีคู่ตัวเลขซ้ำกัน และเรียงลำดับตัวเลขในแต่ละคู่ได้

การออกแบบอัลกอริทึม (1)

- กำหนดโครงสร้างของอัลกอริทึม
  - การเรียงลำดับอาร์เรย์ของตัวเลขเพื่อให้การค้นหาเป็นไปอย่างรวดเร็ว
  - ตัดสินใจว่าจะใช้ขั้นตอนวิธีใด
    - การเรียกใช้ซ้ำ
    - การใช้ตัวชี้ (pointers)
    - การใช้โครงสร้างข้อมูล hashmap
- ออกแบบลอจิกของอัลกอริทึม
  - ใช้การวนซ้ำสองชั้นเพื่อหาคู่ตัวเลขที่มีผลรวมตรงกับ targetSum
  - ปรับปรุงอัลกอริทึมโดยการลดจำนวนการเข้าถึงข้อมูล
  - โดยใช้ hashmap เพื่อเก็บค่าตัวเลขที่เหลือ

การออกแบบอัลกอริทึม (2)

- เขียนโค้ดสำหรับอัลกอริทึม
  - เขียน **pseudo code** แสดงขั้นตอนวิธีที่ออกแบบการทำงาน
  - ตรวจสอบโค้ดที่เขียนว่าจัดการกับข้อกำหนดและเงื่อนไขที่ได้วิเคราะห์หรือไม่
- ทดสอบอัลกอริทึม
  - ทดสอบอัลกอริทึมด้วยชุดข้อมูลทดสอบที่แตกต่างกัน เพื่อดูว่าโค้ดที่เขียนนั้นทำงานได้ถูกต้องตามความต้องการหรือไม่
  - ใช้ชุดข้อมูลที่มีขนาดใหญ่เพื่อทดสอบประสิทธิภาพและระยะเวลาที่ใช้ในการประมวลผล

pseudo code

```
Function findPairsWithSum(numbers, targetSum):
Define numberMap to be a new HashMap
Define result to be a new List
For each number in numbers do:
Define complement to be targetSum - number
        If numberMap contains key complement then:
                Add the pair [number, complement] to result
        EndIf

        Add number to numberMap with value True
EndFor
Return result
EndFunction
```

JavaScript

```
function findPairsWithSum(numbers, targetSum) {
const numberMap = {};
const result = [];
numbers.forEach(number => {
      const complement = targetSum - number;
    if (numberMap[complement]) {
       result.push([number, complement]);
    }
    numberMap[number] = true;
});
return result;
}
```

3) การพิสูจน์อัลกอริทึม

- ทดสอบกรณีฐาน (Base Cases Testing)
- ทดสอบกรณีตัวอย่าง (Sample Cases Testing)
- ทดสอบกรณีพิเศษ (Special Cases Testing)
- ทดสอบประสิทธิภาพ (Performance Testing)
- การทดสอบขอบเขตข้อมูล (Boundary Cases Testing)
- การทดสอบการทำงานผิดปกติ (Edge Cases Testing)
- การพิจารณาและการแก้ไข

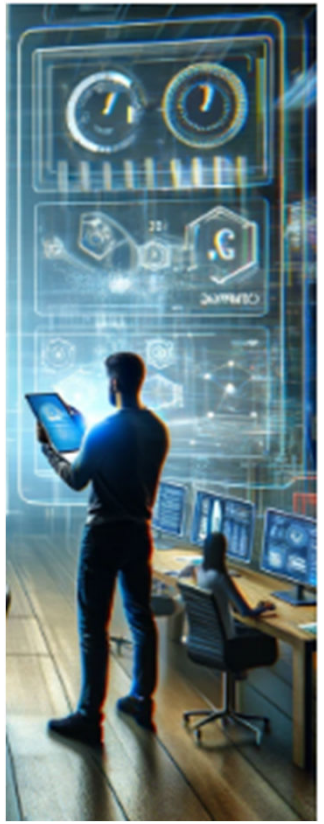
ทดสอบกรณีพื้นฐาน

- การทดสอบความถูกต้องของอัลกอริทึมในสถานการณ์ที่ง่ายที่สุด
  - อาร์เรย์ว่าง (Empty Array)
  - อาร์เรย์ที่มีตัวเลขหนึ่งตัว (Single Element Array)
  - ไม่มีคู่ตัวเลขที่ได้ผลรวมตามเป้าหมาย (No Pairs Matching Target Sum)
  - มีคู่ตัวเลขหนึ่งคู่ที่ได้ผลรวมตามเป้าหมาย (One Pair Matching Target Sum)

```
const numbers = [1, 2, 3, 4, 5]; const targetSum = 5;
console.log(findPairsWithSum(numbers, targetSum)); // [[2, 3], [1, 4]]
```

4) การประเมินประสิทธิภาพ

- ความซับซ้อนด้านเวลา (Time Complexity)
  - ความซับซ้อนด้านเวลาคือ O(n)
  - ชุดข้อมูลที่มีขนาดแตกต่างกัน เช่น 10, 100, 1,000, 10,000
- ความซับซ้อนด้านหน่วยความจำ (Space Complexity)
  - พิจารณาขนาดของโครงสร้างข้อมูล เช่น hashmap หรืออาร์เรย์
- การทดสอบขีดจำกัด (Stress Testing)
  - ทดสอบกับข้อมูลที่ใหญ่ที่สุดในสภาพแวดล้อมการทำงาน
- การวิเคราะห์ผลลัพธ์ (Result Analysis)
  - วิเคราะห์ผลลัพธ์ที่ได้จากการทดสอบ
  - หาทรัพยากรที่จำกัดและพิจารณาปรับปรุงอัลกอริทึม



5) การทดสอบและการดีบัก

- การเขียน Test Cases
  - ชุดการทดสอบที่หลากหลายทุกสถานการณ์
- การทดสอบด้วยโค้ด
  - ใช้โค้ดเพื่อทดสอบอัลกอริทึมตาม test cases ที่กำหนด
- การใช้เครื่องมือดีบัก
  - ใช้ IDE เพื่อกำหนดจุดหยุด (breakpoints) และตรวจสอบการทำงานขณะรัน
  - ค่าตัวแปรและไหลของการคำนวณในแต่ละขั้นตอน
- การแก้ไขข้อผิดพลาด (Error Correction)
  - สาเหตุและแก้ไขปัญหา การปรับโค้ดหรือการปรับโครงสร้างข้อมูล
- การทบทวนโค้ด (Code Review)
  - ให้ผู้อื่นทบทวนโค้ด เพื่อหาข้อผิดพลาด
- การบันทึกผลลัพธ์ (Logging)
  - เก็บรายละเอียดของการทำงาน

6) การปรับปรุงและการคิดค้นใหม่

- ทบทวนผลลัพธ์ที่ได้
- ประเมินลอจิกและโครงสร้างข้อมูล
- ทดสอบความเสถียร
- คิดค้นวิธีการใหม่
- ทดลองและปรับปรุง
- รับความคิดเห็นและการตอบรับ
- ปรับปรุงและทดสอบซ้ำ

สรุปท้ายบท

- แนวคิดเชิงคำนวณเกี่ยวข้องกับการใช้คอมพิวเตอร์เพื่อแก้ไขปัญหาหรือทำงานที่ต้องการ
- การเข้าใจว่าคอมพิวเตอร์ทำงานอย่างไรช่วยให้ดำเนินการได้อย่างมีประสิทธิภาพ
- การพัฒนาอัลกอริทึมเพื่อแก้ไขปัญหาหรือทำงานที่กำหนด
- อัลกอริทึมที่เรียบง่ายหรือซับซ้อน เทคนิคในการออกแบบอัลกอริทึม

ถาม-ตอบ