w13-Lec

Problem Solving

Assembled for 204111

by Kittipitch Kuptavanich

Problem Solving

- ทักษะที่สำคัญที่สุดของ computer scientist คือ ***problem solving***.
  - the ability to formulate problems,
    - ความสามารถในการกำหนดปัญหา
  - think creatively about solutions,
    - การคิดวิธีการแก้ปัญหาอย่างสร้างสรรค์
  - and express a solution clearly and accurately.
    - การแสดงวิธีการแก้ปัญหาอย่างชัดเจน และถูกต้อง

Problem Solving Techniques

- **Abstraction:**
  - แก้ปัญหาจากระบบจำลอง (model) แล้วจึงนำไปใช้กับระบบจริง
    solving the problem in a model of the system before applying it to the real system
- **Analogy:**
  - นำแนวคิด/คำตอบปัญหาที่คล้ายกันมาใช้
    using a solution that solves an analogous problem
- **Brainstorming:**
  - การระดมสมอง ใช้แนวคิด/วิธีการแก้ปัญหาหลาย ๆ วิธี มาผสมผสาน และต่อยอด จนพบคำตอบที่เหมาะสมที่สุด
    (especially among groups of people) suggesting a large number of solutions or ideas and combining and developing them until an optimum solution is found

Problem Solving Techniques [2]

- **Divide and conquer:**
  - แบ่งปัญหาใหญ่ ๆ ที่ยากและซับซ้อนลงเป็นปัญหาย่อย ๆ ที่แก้ไขได้
    breaking down a large, complex problem into smaller, solvable problems
- **Hypothesis testing:**
  - การทดสอบสมมติฐาน: ตั้งสมมติฐาน หรือคำอธิบายของปัญหาที่เป็นไปได้แล้วตรวจสอบโดยการพิสูจน์ (prove หรือ disprove)
    assuming a possible explanation to the problem and trying to prove (or, in some contexts, disprove) the assumption
- **Lateral thinking:**
  - คิดนอกกรอบ คิดแนวกว้าง คิดแนวข้าง แก้ปัญหาจากมุมมองที่ต่างออกไปอย่างสร้างสรรค์
    approaching solutions indirectly and creatively

Problem Solving Techniques [3]

- **Means-ends analysis:**
  - ในแต่ละขั้นตอน เลือกการดำเนินการที่เข้าใกล้จุดมุ่งหมายมากขึ้น
    choosing an action at each step to move closer to the goal
- **Method of focal objects:**
  - การนำลักษณะของสิ่งที่แตกต่างกันมารวมให้เกิดสิ่งใหม่ (มีหินหนึ่งก้อน ทำเป็นสีวาดรูปได้อย่างไร)
    synthesizing seemingly non-matching characteristics of different objects into something new
- **Morphological analysis:**
  - พิจารณาความเกี่ยวข้องและปฏิสัมพันธ์ของแต่ละส่วนโดยรวม
    assessing the output and interactions of an entire system
- **Proof:**
  - พยายามพิสูจน์ว่าปัญหานั้น ๆ ไม่มีทางแก้ หากไม่สำเร็จ ⟶ เริ่มต้นการแก้ปัญหา
    try to prove that the problem cannot be solved. The point where the proof fails will be the starting point for solving it

Problem Solving Techniques [4]

- **Reduction:**
  - แปลงปัญหาให้กลายเป็นอีกปัญหาที่มีทางแก้
    transforming the problem into another problem for which solutions exist
- **Research:**
  - ใช้แนวคิดที่มีอยู่หรือ ปรับใช้วิธีแก้จากปัญหาที่มีลักษณะคล้ายคลึงกัน
    employing existing ideas or adapting existing solutions to similar problems
- **Root cause analysis:**
  - พิจารณาหาสาเหตุของปัญหา
    identifying the cause of a problem
- **Trial-and-error:**
  - การแก้ปัญหาแบบลองผิดลองถูก
    testing possible solutions until the right one is found

Problem-Solving with Programming

1. First, you have to understand the problem.
   - เข้าใจปัญหา
2. After understanding, then make a plan.
   - วางแผน
3. Carry out the plan.
   - ทำตามแผน
4. Look back on your work. How could it be better?
   - ตรวจสอบผลที่ได้ ตลอดจนกระบวนการในการแก้ปัญหา ว่ามีสิ่งที่ควรปรับปรุงหรือไม่

Step 1: Understand the Problem

- กำหนดปัญหาให้ชัดเจน
- ไม่ควรข้ามขั้นตอนนี้
  - มักถูกมองข้ามเนื่องจาก อาจเห็นว่าชัดเจนอยู่แล้ว
- <u>คำถามที่ควรถาม</u>
  - โจทย์ให้หาอะไร หรือให้แสดง/พิสูจน์อะไร
  - ลองอธิบายปัญหาด้วยถ้อยคำของตัวเองตามความเข้าใจได้ไหม
  - สามารถวาดรูป หรือไดอะแกรม เพื่อทำให้เข้าใจมากขึ้นได้ไหม

Step 1: Understand the Problem [2]

- คำถามที่ควรถาม (cont):
  - ณ ขณะนี้ มีข้อมูลเพียงพอที่จะหาทางแก้ปัญหาหรือไม่
  - เข้าใจคำทุกคำที่อยู่ในโจทย์หรือไม่
  - จำเป็นต้องถามคำถาม หรือหาข้อมูลอะไรเพิ่มหรือไม่ ก่อนที่จะหาคำตอบได้

Step 2: Devise a Plan

แก้ปัญหาให้ได้ก่อนโดยไม่ต้องใช้การเขียนโปรแกรม

1. ใช้กลยุทธ์การแก้ปัญหาที่กล่าวมา
   - **Abstraction, analogy, reduction, trial-and-error, proof, etc...**
2. พิจารณาทางแก้ปัญหาหลาย ๆ ทาง
   - คุณสมบัติที่ต้องการ: ความชัดเจน ความไม่ซับซ้อน ประสิทธิภาพ ความสามารถในการนำไปประยุกต์ใช้กับกรณีอื่น ๆ
   - Future concerns (after more CS courses): usability, accessibility, security, privacy, testability, reliability, scalability, compatibility, extensibility, durability, maintainability, portability, provability, ...

Step 2: Devise a Plan

3. เขียนวิธีแก้ปัญหาในลักษณะที่สามารถนำไปแปลงเป็น code ได้สะดวก
   - ใช้ขั้นตอนที่สั้นและชัดเจน ไม่จำเป็นต้องอาศัยการตัดสินใจ หรือการตีความเพิ่มเติม
   - ทำให้ตอนการแก้ปัญหาดังกล่าว ใช้ได้กับ input ขนาดใหญ่
   - ไม่ต้องอาศัยความจำของผู้อ่านในการแก้ปัญหา
     - เขียนออกมาให้ชัดเจนว่าจำเป็นต้องจำอะไรบ้าง
     - แล้วตั้งชื่อสิ่งเหล่านั้นให้สื่อความหมาย (เพื่อนำสิ่งที่ต้องจำเหล่านั้นนำมาแปลงเป็น variable เมื่อต้อง code)

Step 3: Carry out the Plan

Translate your solution into code

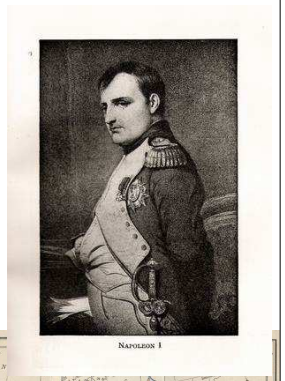
1. เขียน test case (ควรทำเป็นสิ่งแรก ไม่ใช่สิ่งสุดท้าย)
2. แปลงวิธีการแก้ปัญหา จาก Step 2 ทีละขั้นตอน ให้เป็น code
   - ใช้วิธีแบ่งปัญหาจากปัญหาใหญ่เป็นปัญหาเล็ก top-down design (Divide and Conquer)
3. ทดสอบวิธีแก้ปัญหา (robotically and manually)
   - หากรณีขอบเขต และ กลุ่ม (class) ของ input
   - มองหา case ยกเว้นต่าง ๆ เช่น หารด้วย 0 หรือ input มีขนาดใหญ่มาก ๆ หรือเล็กมาก ๆ

Step 4: Examine and Review

- พิจารณาวิธีแก้ปัญหาที่ได้ด้วย **common sense**
- อภิปราย/พิจารณา/วิพากษ์ ข้อดีข้อเสียของ **solution** ที่ได้กับคนอื่น ๆ
  - ไม่แนะนำให้ทำกับ **assignment** ที่ต้องทำเป็นงานเดี่ยว
- เก็บ **solution** ไว้ในที่ ๆ หาได้สะดวก

Divide and Conquer

- หรือ **Divide and Rule**
  - ในทางประวัติศาสตร์และการปกครอง คือการสร้างอำนาจหรือรักษาอำนาจไว้โดยวิธี
    - แบ่งเป้าหมายเป็นหน่วยเล็ก ๆ ที่มีกำลังน้อยกว่า **(Divide)**
    - แล้วเข้ายึดอำนาจทีละส่วน **(Conquer)**



Divide and Conquer [2]

- การแก้ปัญหาโดยใช้หลัก **Divide and Conquer** มี 3 ขั้นตอน
  - **Divide** แบ่งปัญหาที่ต้องการแก้ เป็นปัญหาย่อย **(Subproblem)** - ควรแบ่งแล้วปัญหาเล็กลงหรือซับซ้อนน้อยลง
  - **Conquer** แก้ปัญหาย่อย
  - **Combine** นำคำตอบของปัญหาย่อยมารวมกันเพื่อให้ได้คำตอบของปัญหาหลัก

Top-Down Design (Divide-and-Conquer)

- เขียน **function** จากใหญ่ไปเล็ก
  Write functions top-down
  - สมมติว่า **helper function** ต่าง ๆ เขียนเสร็จแล้ว
    Assume helper functions already exist!
- ทำการ **test function** จากเล็กไปใหญ่
  Test functions bottom-up
  - ไม่นำ **function** ใด ๆ มาเรียกใช้ จนกว่าจะผ่านการ **test** อย่างถี่ถ้วน
    Do not use a function before it has been thoroughly tested
- ในทางปฏิบัติสามารถเขียน **stub function** มาใช้ชั่วคราวขณะทำ **top-down design**
  Practicality: May help to write stubs (simulated functions as temporary placeholders in top-down design)

Reference

- http://en.wikipedia.org/wiki/Problem_solving
- http://en.wikipedia.org/wiki/How_to_solve_it
- http://www.kosbie.net/cmu/spring-12/15-112/handouts/notes-problem-solving.html