นายวศิน วิวงศ์ศักดิ์ 5131054521
นายวาทิศ เปี่ยมสัมฤทธิ์ 5131055121

NP/NP-Hard Problems

นิยามอย่างง่ายๆของคำว่า “ปัญหายาก” คือ ปัญหาที่คอมพิวเตอร์ต้องเสียแรงเยอะในการแก้ นิยามอย่างเป็นทางการ เป็นดังนี้

- ปัญหาง่าย : ปัญหาที่ใช้เวลา O(T(x)) ในการแก้ โดย T(x) เป็น polynomial
- ปัญหายาก : ปัญหาอื่นๆนอกเหนือจากนั้น

Unsolvable/Undecidable Problem

ตัวอย่างเช่น Halting Problem (ต้องการหาว่าโปรแกรมที่รันจะติด loop infinite หรือไม่)

Input : program P และ input S

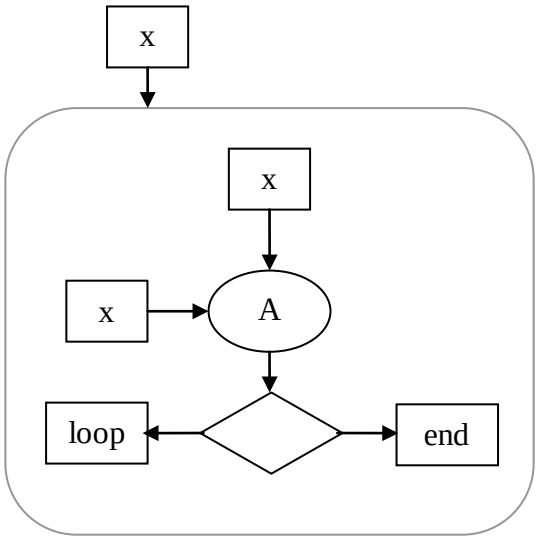
Output : “Yes” ถ้าโปรแกรมทำงานได้จบ, “No” ถ้าโปรแกรมติด loop infinite

ถ้าจะมี algorithm จะต้องเป็น algorithm ที่สามารถตอบได้ทุกๆ instance ของปัญหา ซึ่งมีคนพิสูจน์ไว้แล้วว่าไม่มี algorithm (“Halting is undecidable”) ที่ตอบปัญหา Halting Problem ดังกล่าวได้ ดังนี้

<u>Proof By contradiction</u>

สมมติให้ A(P,S) เป็นฟังก์ชันที่ตอบปัญหา Halting Problem… พิจารณาโปรแกรม kb.cpp ต่อไปนี้

```
bool Kaboom(x){
        If(A(x,x)){
                while(true) do;
        } else{
                printf("finish!");
        }
}
```



จะเห็นว่าถ้าเราเรียก Kaboom(kb.text); จะแยกคิดได้เป็น 2 กรณี ดังนี้

Case I : A(kb.cpp, kb.cpp) ตอบ Yes(โปรแกรม kb.cpp ทำงานได้จบ) โปรแกรมจะเข้าไปทำงานใน then clause ซึ่งเป็นการวน loop while true นั่นคือโปรแกรมจะติด infinite loop เกิดข้อขัดแย้ง

Case II : A(kb.cpp, kb.cpp) ตอบ No(โปรแกรม kb.cpp ติด loop infinite) โปรแกรมจะเข้าไปทำงานใน else clause ซึ่งจะแสดงคำว่า finish! ออกมา นั่นคือ โปรแกรมทำงานได้จบ เกิดข้อขัดแย้ง

สรุป โปรแกรม kb.cpp เป็นตัวอย่างหนึ่งของโปรแกรมที่ไม่สามารถตอบได้ว่าติด infinite loop หรือไม่

มีคน Proof แล้วว่า "ไม่มีทางที่เราจะคิดระบบคณิตศาสตร์ที่ไม่มีปัญหาที่เราคิดไม่ออกได้" หรือแปลอีกนัยหนึ่งว่า "คณิตศาสตร์ไม่สมบูรณ์ในตัวของมันเอง"

Hard vs Easy

ปัญหาไหนยาก ปัญหาไหนง่าย เราเปรียบเทียบโดยความยากในมุมของ "คอมพิวเตอร์" (คอมพิวเตอร์ใช้เวลาแค่ไหนในการทำงานเพื่อให้ได้คำตอบออกมา"

จะเทียบว่าปัญหา 2 ปัญหา A B ใดๆ อันไหนง่ายกว่าเราจะดูจากสิ่งที่เราเรียกว่า "Reducability"

Reducability

ความสามารถในการลดรูปได้ คือ การแปลงจากปัญหาหนึ่งไปเป็นอีกปัญหาที่เราสามารถแก้ได้แล้วอาศัยความสัมพันธ์ที่ได้ย้อนกลับมาหาคำตอบให้ปัญหาเริ่มต้น ตัวอย่างเช่น

ปัญหาการหาจำนวนน้อยสุดอันดับที่ 5 *แปลงเป็น* ปัญหาการเรียงลำดับจำนวนทั้งหมด

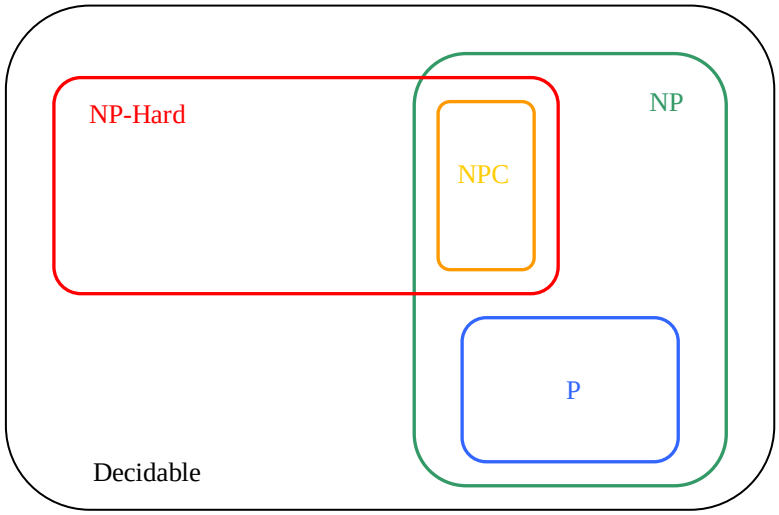
<u>คำถาม</u> ถ้าเราสามารถแปลงจากปัญหา A ให้เป็นปัญหา B ได้ ปัญหาไหนจะยากกว่ากัน ?

<u>ตอบ</u> $A \leq B$ (B ไม่ง่ายไปกว่า A)

ถ้าแก้ปัญหา B ได้ ก็จะสามารถแก้ปัญหา A ได้เช่นกัน

- ถ้าสามารถสร้างตึก 20 ชั้นได้ ก็สามารถสร้างตึก 2 ชั้นได้อย่างไม่ยาก
- คูณง่ายกว่าบวก เพราะถ้าทำบวกได้ การคูณก็คือการบวกซ้ำๆ นั่นเอง
- การเรียงลำดับยากกว่าการหาจำนวนน้อยสุดอันดับที่ 5 เพราะการเรียงลำดับจะทำให้รู้ค่าทุกอันดับเลย

Types of problem



Decision problem (ครอบคลุมทุกปัญหาในโลก)

คือ ปัญหาที่มี output เป็น Yes/No โดยเราสามรถแปลงทุกๆปัญหาให้มาเป็นปัญหาแบบ Decision problem ได้

P problem

ปัญหาง่ายๆที่ใช้เวลาในการแก้เป็น polynomial time หรือดีกว่า

เช่น ปัญหา Sorting, MSS, MST

NP problem

นิยาม 1 : ปัญหาที่สามารถแก้ได้ในเวลา exponential time หรือดีกว่า

นิยาม 2 : ปัญหาที่สามารถทวนสอบได้(ให้คำตอบและหลักฐานมา จะต้องตรวจสอบได้ว่าจริงหรือไม่)ในเวลา polynomial time

นิยาม 3 : ปัญหาที่สามารถหาคำตอบได้ด้วย non-deterministically provable [วิชา formal language]

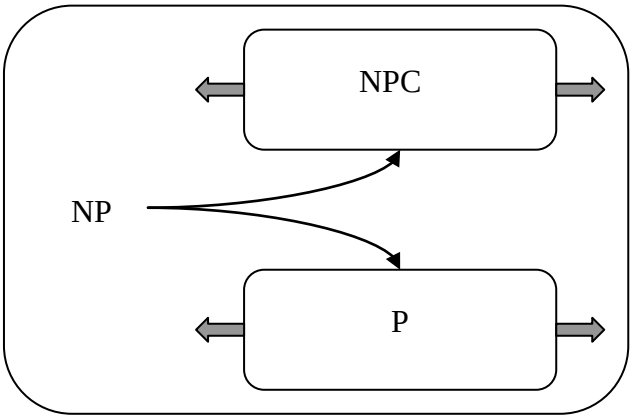
เช่น ปัญหา Graph isomorphism และ ปัญหาทุกๆอย่างที่เคยเรียนมา

NP-Hard problem

ปัญหาที่ทุกๆปัญหาใน NP สามารถแปลงแบบ polynomial มายังปัญหานี้ได้

NPC problem

ปัญหา NP-Hard ที่อยู่ในส่วนของ NP (intersection กันนั่นเอง)

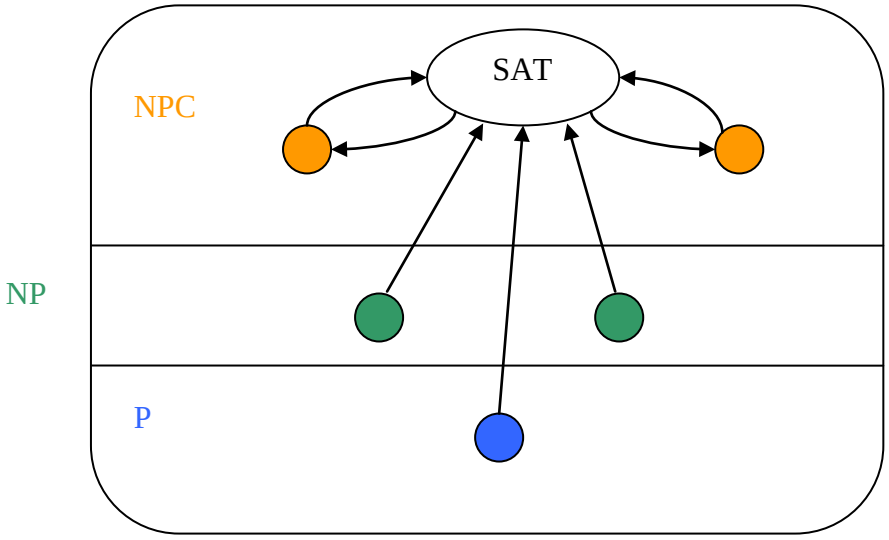


ปัญหา NP จะถูกพิสูจน์กลายเป็นปัญหา NPC หรือ P ทำให้ ส่วนของปัญหา NPC และ P มีขนาดใหญ่ขึ้นเรื่อยๆ

นายวศิน วิวงศ์ศักดิ์ 5131054521
นายวาทิศ เปี่ยมสัมฤทธิ์ 5131055121

Cook's Theorem

ทุกๆปัญหาใน NP problem สามารถ reduce เป็น SAT problem (ปัญหาที่คิดออกด้วยวงจรไฟฟ้า) ได้



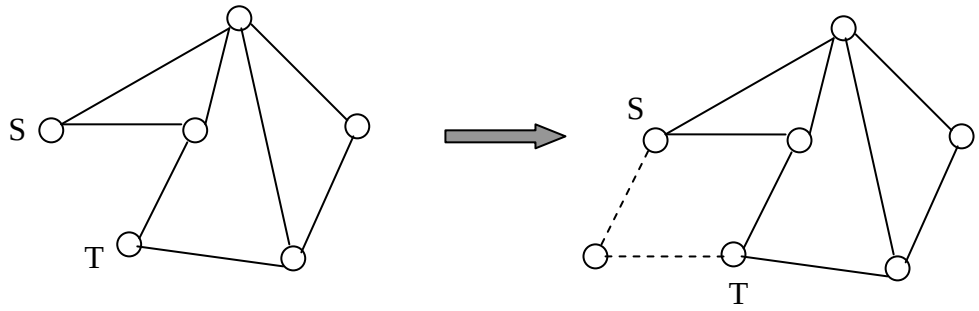
ขั้นตอนเมื่อเจอปัญหาใหม่ๆ

1.) ดูว่าเป็นปัญหาที่ง่ายไหม --> P problem
2.) ดูว่ามีปัญหาจาก NPC ที่ reduce มาได้หรือไม่ --> NPC problem

How to reduce the problem

- Hamiltonian path >> Hamiltonian cycle

นายวศิน วิวงศ์ศักดิ์ 5131054521
นายวาทิศ เปี่ยมสัมฤทธิ์ 5131055121



Graph A **Graph B**

ถ้า B มี cycle แล้ว A ก็จะสามารถหา path

ถ้า B ไม่มี cycle แล้ว A ก็จะไม่มี path (พิสูจน์มาจาก contrapositive ถ้า A มี path แล้ว B จะมี cycle)

- SAT >> 3SAT

  $(a_1 \vee a_2 \vee a_3 \vee \ldots \vee a_k)$

  แปลงเป็น

  $(a_1 \vee a_2 \vee x_1)(\sim x_1 \vee a_3 \vee x_2)(\sim x_2 \vee a_4 \vee x_3)\ldots(\sim x_{k-2} \vee a_k \vee x_{k-1})$