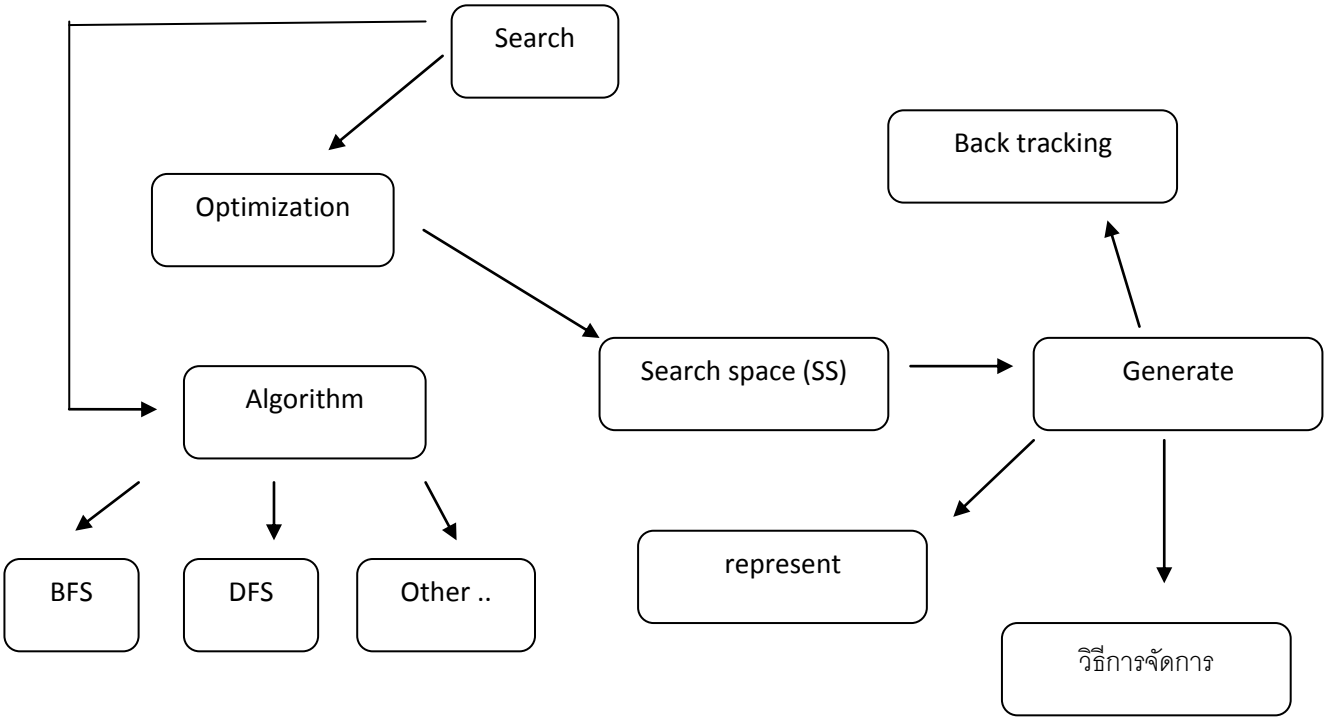
Lecture note 1/9/2010

สิ่งที่จะเรียนหลักๆ



Optimization problem เป็น class ของ ปัญหา

ปัญหาแทบทั้งหมดสามารถแปลงเป็น Optimization problem ได้

ปัญหาของ Optimization problem สามารถอธิบายได้ด้วย function F(s)

- A domain of s must also be describe

  Called D, the set of possible solution.

- The solution of the instance is

S such that F(s) is maximized (or min)

Idea : ลองทุกๆ possibility ที่เป็นไปได้และใช้ตัวที่ดีที่สุด (ตัวที่ต้องการ มักเป็น max or min) เป็นคำตอบ แต่การทำเช่นนี้มีข้อเสียคือ ช้ามาก เพราะ ต้องลองทุกวิธี นั่นคือ O(D) ซึ่ง D คือ size ของ domain

Example 1 : find max in a array.

D คือ set of all indexs

F(s) คือ function ที่ให้ค่าใน index

และดูว่าค่าที่ได้จาก F(s) อันไหนมากสุดนั่นคือคำตอบที่เราต้องการ

Example 2 : shortest path

D: set of all path

F(s): เป็น function ที่ให้ผลลัพท์คือ ระยะทางของ path ที่เป็น input

และดูว่าค่าที่ได้จาก F(s) ค่าไหนมีค่าน้อยสุด นั่นคือคำตอบ

Example 3: Sorting

D: permutation ทั้งหมดของ input เช่น ถ้า input คือ a1,a2,a3. ดังนั้น

D = {(a1,a2,a3),(a1,a3,a2),(a3,a1,a2),(a3,a2,a1),(a2,a1,a3),(a2,a3,a1)}

F(s) จะให้ผลลัพท์คือ จำนวนคู่ของ input ที่ไม่เรียงค่า

ดังนั้นสิ่งที่เราจะสนใจคือ input ที่ทำให้ ผลลัพท์จาก F(s) มีค่า 0 นั่นคือไม่มีคู่ใดเลยไม่ได้เรียงค่า

แต่ที่น่าสังเกตคือ ถ้าใช้วิธีนี้ จะได้ O(n!) ซึ่ง n คือจำนวนของข้อมูลทั้งหมด ซึ่งวิธีนี้ช้ามาก ดังนั้นจึงไม่มีใครทำกัน จึงมีวิธี sort แบบต่างๆที่เราได้เรียนกันไปแล้วใน data structure

Example 4: maximum sum of subsequence

D: subsequence ทั้งหมดที่เป็นไปได้

F(s) : ผลรวมของ subsequence

เลือกผลลัพท์จาก F(s) ที่มีค่ามากที่สุดมาเป็นคำตอบ

Example 5: closest pair

D: ทุกคู่จุดที่เป็นไปได้

F(s): distance ระหว่างคู่จุดที่เป็น input

Input ที่ทำให้ ค่าจาก F(s) มีค่าน้อยที่สุด นั่นคือ คำตอบ

Example 6: Longest common subsequence

D: ทุกๆ subset ที่เป็นไปได้ของตัวที่มีขนาดสั้นกว่า

F(s) : รับ subsequence และดูว่ามีในอีก string รึเปล่า ถ้ามีก็ให้ตอบความยาวออกมา

Input ของ F(s) ที่ทำให้ F(s) มีค่ามากสุด นั่นคือ คำตอบ

State Space Search (SSS)

Search space มีได้หลายแบบขึ้นกับ F(s) ถ้า F(s) ดีจะได้ search space ที่เล็กซึ่งดี แต่ถ้า F(s) ไม่ค่อยดี จะได้ F(s) ที่ค่อนข้างใหญ่ซึ่งไม่ค่อยดี (เกินความจำเป็น)

Example: find max in array

Search space คือ ทุกๆ index ใน array

Example : maximum sum of subsequence

Search space : คู่ของจุดเริ่มและจุดสุดท้ายของ element

Generate : 1. เลือกตำแหน่งของตัวแรก (หัว)

2. เลือกตำแหน่งของตัวสุดท้าย

Check each solutions : บวกค่าแต่ละ solution จากจุดเริ่มต้นจนถึงจุดสุดท้าย และดูว่า solution ไหนให้ค่ามากสุด นั่นคือคำตอบ

สามารถเขียน code ในส่วนของการ generate ได้ดังนี้

```
// gen

int max = - infinity;

for(int I = 0;i<n;i++){

        for( int j = i + 1;j<n;j++){

                int sum = evaluate(i,j);

                if(sum > max) max = sum;

        }

}

void evaluate(int I,int j){

        int sum = 0;

        for(int k = i; k < j;k++){

                sum  += A[k];

        }

return sum;
```

search space ควรที่จะครอบคลุมเฉพาะ possibility ที่เป็นไปได้ ไม่งั้นอาจจะเกิดปัญหาขึ้น ดังที่จะแสดงต่อไป

// gen

code ต่อไปนี้แสดงการสร้าง search space ที่มากเกินไป ซึ่งก่อให้เกิดปัญหาได้

```
int max = - infinity;

for(int i = 0;i<n;i++){

        for( int j = 0;j<n;j++){                // แก้จาก code เดิมคือ  ให้ j = 0  ดังนั้นจะมีคู่ที่ไม่ควรจะมีเกิดขึ้น

                int sum = evaluate(i,j);

                if(sum > max) max = sum;

        }

}

void evaluate(int I,int j){

        int sum = 0;

        for(int k = i; k < j;k++){

                sum  += A[k];

        }

return sum;
```

จาก code ด้านบน เมื่อมี คู่ที่ไม่ควรจะมีเกิดขึ้น ใน for ของ evaluate จะสามารถแก้ปัญหาได้ถ้าดูผิวเผิน แต่จะเกิดปัญหาเช่นในกรณีนี้ ถ้า [-1,-2-,3] ซึ่งค่าที่ evaluate return กลับไปนั้น คือ 0 นั่นคือจะผิดจากความเป็นจริงที่ควรจะเป็น จากตัวอย่างนี้แสดงให้เห็นว่า การเลือก search space ที่มากเกินไปอาจก่อให้เกิดปัญหาได้คือ แก้ evaluate นิดหน่อย ดังจะแสดง

```
void evaluate(int I,int j){

        if(i > j) return – infinity;   // บรรทัดนี้จะแก้ปัญหาที่กล่าวมาข้างต้นได้

        int sum = 0;

        for(int k = i; k < j;k++){

                sum  += A[k];

        }

return sum;
```

ถึงแม้ว่าจะแก้ปัญหาได้ แต่เราควรที่จะเลือก search space ที่เหมาะสมตั้งแต่ต้นจะดีกว่า

Example: Minimal spanning tree

วิธีการเลือก input ที่จะป้อนให้กับ F(s) มีได้หลายวิธี เช่น

แบบแรก : เลือกแต่ละ edge ว่าเอาหรือไม่เอา ดังนั้น จำนวน solution ทั้งหมดที่มีคือ 2^n โดย n คือ จำนวน edge ทั้งหมด แต่การเลือกแบบนี้จะต้องมีภาระใน F(s) คือ F(s) จะต้อง ทำงาน 3 อย่างคือ

1. ดูว่าเป็น tree หรือไม่
2. ดูว่า connected หรือไม่
3. บวกค่า weight ทั้งหมดของ edge ที่ส่งเข้ามา

   และสุดท้ายคือ เราต้องการค่าที่น้อยที่สุด ดังนั้นจึงต้องเลือกค่าน้อยสุด

แบบสอง : เลือก subset of edge of size |V| - 1 ดังนั้นจำนวน solution ทั้งหมดที่เป็นไปได้คือ C(E,V-1) การเลือกแบบนี้จะต้องมีภาระใน F(s) คือ F(s) จะต้อง ทำงาน 2 อย่างคือ

1. ดูว่าเป็น tree หรือไม่
2. บวกค่า weight ทั้งหมดของ edge ที่ส่งเข้ามา

   และสุดท้ายคือ เราต้องการค่าที่น้อยที่สุด ดังนั้นจึงต้องเลือกค่าน้อยสุด

การที่เราไม่ต้อง check ว่าเป็น connected หรือไม่ เนื่องจาก ถ้าเป็น tree แล้ว มี V-1 edge ยังไงก็ต้อง connected แน่ๆ

แบบสาม : เลือกทุกๆ edge ใน x โดยใช้ cut property ช่วย ดังนั้น solution ที่เลือกมา เราแน่ใจได้ว่า เป็น tree และ connected ด้วย ดังนั้นในขั้นตอนของ F(s) เราจึงทำแค่เพียง บวก weight ของ edge ทั้งหมดเท่านั้น ไม่ต้อง check คุณสมบัติว่าเป็น tree หรือไม่ หรือ connected หรือไม่

Example : Pachinko รายละเอียดของ Pachinko คือเหมือนกับในข้อสอบ mid-term

D : คือ set ของการเลือกว่าจะเลือก left หรือ right เช่นถ้ามี 3 level จะมี solution ที่เป็นไปได้ทั้งหมดคือ

D = {(LLR),(LRL),(LLL),(RRR),(RLR),(RRL)}

ดังนั้น จะมีจำนวน solution ทั้งหมดที่เป็นไปได้เท่ากับ $O(2^n)$ โดย n คือจำนวน level ของ Pachinko

……..