

NP-COMPLETE Report

วิชา Algorithm Design 2110327

เรื่อง Maximum Common (Induced) Subgraph

จัดทำโดย

นายภาคพน วรรณวณะ 6130391021

Introduction

ปัญหา **maximum common subgraph** เป็นปัญหาที่ว่าด้วยการหา **subgraph** ที่ซึ่งเป็น **isomorphic** ของกันและกัน ด้วยความที่กราฟนั้นเป็นสิ่งที่ใช้ในการจำลองข้อมูล หรือปรากฏการณ์ในชีวิตประจำวันหลายอย่าง จึงเกิดปัญหา **maximum common subgraph** จากการศึกษาโมเดลของสสาร การตรวจหามัลแวร์ การวิเคราะห์โรคได้ และ **computer vision** เป็นต้น

ปัญหา **maximum common subgraph** นั้นถูกจัดอยู่ในประเภท **NP-hard** หมายความว่า เป็นปัญหาประเภทที่ใช้เวลาในการตรวจคำตอบว่าถูกต้องเป็น **polynomial time** และยากไม่น้อยกว่าทุกปัญหาใน **NP** นั่นเอง

โดยในรายงานนี้จะนำเสนอถึงวิธีการแก้ปัญหา **maximum common subgraph** ในรูปแบบของ **induced subgraph** ซึ่งจะสนใจจำนวน "ปม" มากที่สุดที่ทำให้ **induced subgraph** นั้นมีสมบัติ **isomorphic** ต่อกันและกัน **induced subgraph** นั้นหมายความว่าเมื่อเลือกปมกลุ่มใด ๆ ในกราฟที่กำหนด **subgraph** ที่เลือกจำเป็นจะต้องมีเส้นเชื่อมระหว่างปมตามจริงตามกราฟที่กำหนด หมายความว่าไม่สามารถที่จะมี **edge** ที่ตัดทิ้งได้นั่นเอง ในรายงานนี้จะนำเสนอ **algorithm** ที่ชื่อว่า **McSplit algorithm** โดยจะใช้การเปรียบเทียบ **label** ของแต่ละปมที่พิจารณามาจากการเชื่อม หรือไม่เชื่อมของปมที่เลือกกับปมอื่นๆ ซึ่งมีการใช้ **branch and bound** ร่วมด้วยซึ่งจะอธิบายต่อไป

Input - Output

โดยปกติแล้วปัญหานี้เป็นปัญหา **optimization problem** แต่โจทย์ต้องการศึกษา **decision problem** จึงต้องมีการนำ **algorithm** มาดัดแปลงโดยมี **input** และ **output** ดังนี้

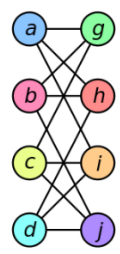
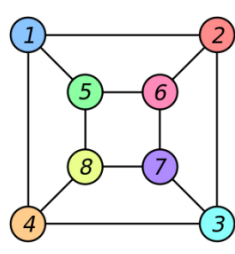
Input เป็นกราฟสองกราฟที่ต้องการจะหา **MCIS** ประกอบด้วย จำนวนปม ชื่อของแต่ละปม จำนวนเส้นเชื่อมระหว่างปม ปมที่เชื่อมกัน ค่า **K**

Output คำตอบ **yes/no**

พิสูจน์ NP

โดยจากalgorithmที่ทำ จะทำการเก็บคู่ปมที่สามารถแทนกันได้ไว้ในเซตMจึงสามารถนำข้อมูลดังกล่าวมาเช็คความ ถูกผิดได้ โดยสมบัติของisomorphic graphคือกราฟคู่ที่สามารถเปลี่ยนvertexกันยังเป็นกราฟเดียวกันโดยหากvและ wเป็นปมในกราฟทั้งgและhตามลำดับ ปมที่เชื่อมกับปมv ในกราฟgจะต้องเป็นคู่vertexที่สามารถแทนกันได้ในกราฟh แปลว่าปมเดียวกันนั้นในhจะต้องเชื่อมกับbด้วยดังรูปจะเห็นว่า aถูกจับคู่กับ 1 และ aเชื่อมกับg, h และi ซึ่งในการจับคู่ของ กราฟHปมgนั้นคู่กับ5 hคู่กับ2และiคู่กับ4 โดยทั้ง5,2,4นั้นเชื่อมไปยัง1เช่นเดียวกัน และเป็นเช่นนี้ไปในทุกปมของกราฟ ทั้งสองจึงใช้หลักการดังกล่าวมาเขียนในภาษาC++

```
bool checkMCIS(set<pair<int,int>> M,int k , map<int,set<int>> NG,
map<int,set<int>> NH){
if(M.size()<k)return false;//หากขนาดน้อยกว่าKที่ต้องการจะผิดทันที
map<int,int> Map;
for(auto x:M) {Map[x.first]=x.second;Map[x.second]=x.first;}
for(auto x:M){
int v=x.first;
int w=x.second;
for(auto y:NG[v]){
if(Map.find(y)!=Map.end()){
if((NH[w].find(Map[y])==NH[w].end())){
return false;
}
}
}
for(auto y:NH[w]){
if(Map.find(y)!=Map.end()){
if((NG[v].find(Map[y])==NG[v].end())){
return false;
}
}
}
}
return true;
}
```

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

ซึ่งมีTime complexity เป็น $O(n^2 \log n)$ เป็นpolynomial time จึงจัดอยู่ในNP problemนั่นเอง

พิสูจน์ NP-COMPLETE

ปัญหา maximum common subgraph นั้น เป็นปัญหาที่สามารถ ลดรูปมาจาก

ปัญหา clique problem ซึ่งเป็นปัญหาที่ว่าด้วยการหา subgraph ขนาด k ที่ทุกจุดใน clique นั้นเชื่อมต่อถึงกัน
หมด ซึ่งปัญหานี้ถูกจัดอยู่ใน NP-hard

โดยเริ่มจากกำหนดให้กราฟเริ่มต้นของ maximum common subgraph เป็นกราฟ $G(v_1, e_1)$ และ $H(v_2, e_2)$
ซึ่งมี v_1' และ v_2' ขนาด k เป็น pm ของ subgraph ที่เป็น isomorphic กัน

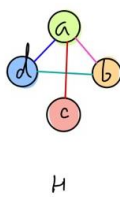
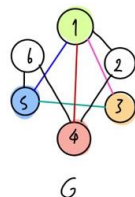
และในปัญหา clique problem ของกราฟ $C(v, e)$ มี goal เป็นขนาด g และจะต้องการ subgraph ที่มีขนาด g และเป็น
complete graph ใน C เริ่มการลดรูปจากการกำหนดกราฟ C' เป็น complete graph ที่มี pm เหมือนกับ C
หมายความว่าเมื่อเลือก pm ชุดใดๆ จะเป็น clique เสมอ นำกราฟ C และ C' มาเป็นกราฟเริ่มต้นของ maximum
common subgraph โดยสมมติว่ามีเซต $v_1' \subseteq C$ และ $v_2' \subseteq C'$ ที่เป็นเซตของ pm คำตอบของปัญหา ซึ่งเป็น
คำตอบเดียวกับคำตอบของปัญหา maximum clique in graph C ซึ่งเป็น NP-hard

และการแปลงของปัญหานี้คือการเพิ่ม edge ให้กับทุกคู่ของจุดซึ่งไม่ได้เชื่อมกัน ใช้เวลาเป็น $O(n^2)$ เมื่อ n เป็นจำนวน pm ใน
กราฟ C จึงสรุปได้ว่า ปัญหา maximum common subgraph เป็นปัญหา NP-Hard และ จากข้อแรกเป็น
ปัญหา NP ดังนั้นจึงจัดอยู่ในปัญหา NP-complete

McSplit Algorithm

เริ่มจากการกำหนดให้กราฟของเรานั้นเป็น undirected graph without loop $G(V_G, E_G)$ และ $H(V_H, E_H)$ เป็น input graph การหา maximum common induced subgraph ด้วย algorithm นี้ จะได้ผลลัพธ์เป็นเซตของคู่ปมที่เลือกให้แทนที่กันในการทำ isomorphic $M = \{ \{v_1, w_1\}, \{v_2, w_2\}, \dots, \{v_i, w_i\} \}$ ซึ่งขนาดของ M จะบ่งบอกถึงจำนวน vertex ที่เป็นคำตอบ

เริ่มแรกนั้น M จะรับมาเป็นเซตว่าง (\emptyset) และเพิ่มขึ้นเรื่อยๆ จากการทำ depth first search โดยในการหาแต่ละระดับนั้น จะทำการเลือก ปม v ในกราฟ G มา จากนั้น เซต M จะถูกเพิ่มคู่ปม $\{v_i, w_i\}$ ลงไป หรือ จะไม่จับคู่ปม v และนำ v ออกจากการ ค้นหาแล้วกระทำในระดับต่อไป ในการทำแต่ละระดับนั้นเมื่อเลือกปม v แล้วจะวนทำทุกค่า w ที่เป็นปมที่มีเส้นเชื่อมมาจาก v และถือว่า v, w เป็นปมที่จะถูกเลือกไปใส่ใน M แล้วนั่นเอง ซึ่งจะถูกใส่ได้ต่อเมื่อมี label ของปมเหมือนกันเท่านั้น (การเลือก ครั้งแรกนั้นทุกปมมี label เท่ากันเสมอ) label ในแต่ละปมเป็นการจัดกลุ่มให้ ปมที่เชื่อมกันมี label เป็น 1 และไม่เชื่อมกัน มี label เป็น 0 เมื่อทำการเลือก v, w แล้วก็จะให้นำปมที่เหลือใน subgraph มาทำการ label ซึ่งการ label นี้จะเพิ่มไปใน ทุกๆ ระดับ (ยังเก็บ label ก่อนหน้าไว้) ยกตัวอย่างกราฟดังรูป ในระดับแรกเลือก $v=1, w=a$ จากนั้นในกราฟ G จะ label ปมที่เชื่อมกับ 1 เป็น 1 และไม่เชื่อมเป็น 0 ส่วนกราฟ H จะดูที่ปม a ในทำนองเดียวกัน ระดับที่สองเลือกคู่ปมที่มี label เหมือนกัน 3 และ b และทำการ label ปมที่เหลือในทำนองเดียวกันแต่จะสังเกตได้ว่าไม่รวมปมที่เลือกไปแล้ว (1, 3, a, b) ใน ขั้นนี้จะสังเกตได้ว่า 6 มี label ที่ไม่มีคู่กับปมใดเลยในกราฟ H จึงไม่มีประโยชน์ที่จะติด label ในระดับต่อไปจึงตัดออกไปได้ เลย ระดับที่สามเลือกคู่ปมที่มี label เหมือนกัน 4 และ c และทำการ label เหมือนเดิมจะได้คู่สุดท้ายเป็น 5 และ d ส่วนปม 2 ถูกตัดทิ้งและจบการทำงาน



$$M = \{ \{1, a\}, \{3, b\}, \{4, c\}, \{5, d\} \}$$

1: $v=1, w=a$			
label G		label H	
2	1	5	1
3	1	6	0
4	1		
2: $v=3, w=b$			
label G		label H	
2	1		
4	1	c	1
5	1	d	1
6	0		
3: $v=4, w=c$			
label G		label H	
2	1		
5	1		
		d	1

จะสังเกตเห็นได้ว่า การเลือกเซต M แบบที่กำหนดเป็นเพียงการเลือกแบบหนึ่งเท่านั้น หากเลือกคู่ปมที่มี **label** เหมือนกันต่างออกไปจากตัวอย่างทำให้เราได้ค่า M ที่ต่างออกไป จึงจำเป็นต้องลองทำทุกๆคู่ที่มี **label** เหมือนกันเพื่อหาเซต M ที่มีจำนวนสมาชิกสูงสุดนั่นเอง หลักในการเลือก V นั้นกำหนดไว้ในฟังก์ชัน **selectVertex** ที่จะเลือก V ที่มีดีกรีมากที่สุดเพื่อเป็นการเพิ่มความน่าจะเป็นที่จะทำ **depth first search** จนถึงความลึกมากที่สุดได้ก่อน และจะทำการเลือกทำ **label** ที่เหมือนกันจากเซตของ **label** ที่เหมือนกันที่มีค่าน้อยที่สุดเมื่อพิจารณาจากทุกค่ามากที่สุดของจำนวนปมในกราฟ G และ H ที่มี **label** นั้นๆในฟังก์ชัน **selectlabelclass** เพื่อประสิทธิภาพที่เร็วขึ้น

เนื่องจากการ **label** นั้นไปได้เพียงสองทิศทางคือ **1** และ **0** จึงไม่จำเป็นต้องเก็บเป็นตัวเลข ใน **algorithm** นี้จะทำการเก็บ **label** ของแต่ละปมในแต่ละการเรียกไว้ในตัวแปร **future** ซึ่งเป็นเซตของ **pair** ของเซต ที่ซึ่งเซตใน **pair** นั้น โดย **pair.first** เก็บปมในเซต G และ **pair.second** เก็บปมในเซต H ที่มี **label** เดียวกันโดยเริ่มแรกจะส่งทุกปมใน G และ $H(\{G, H\})$ มาเพราะถือว่ายังไม่มี **label** เหมือนกันทั้งหมด

หลังจากทำการค้นหาแบบที่มี V เป็นปมหนึ่งทีเลือกแล้ว จะทำการค้นในแบบที่ **ไม่เลือก V** ด้วย โดยจะทำการนำ V ออกจากเซตของปม $V(G)$ และนำ V ออกจากเซต **future** ที่เคยมี V อยู่และจึงกระทำในระดับต่อไปนั่นเอง

อีกหนึ่งประเด็นที่จำเป็นจะต้องแก้ไขคือ **algorithm** ที่นำมาแก้ไขปัญหานั้นเป็นลักษณะของ **optimization problem** ที่ให้ผลลัพธ์เป็นเซต M ที่เก็บคู่ปมคำตอบจำนวนมากที่สุด จึงจะต้องนำมาเปรียบเทียบกับค่าของ k ว่าถ้าหากขนาดของเซต M สูงสุดที่เคยหาเจอ มีค่ามากกว่าหรือเท่ากับ k จะให้ผลลัพธ์เป็น **yes** และให้ **no** หากมีค่าน้อยกว่า k ซึ่งจะอยู่ในรูปแบบของ **decision problem** ดังที่โจทย์นั้นต้องการจะหา

Branch and bound , Backtracking

Bound เป็นค่าที่เรากำหนดไว้เป็น **base case** ซึ่งมีค่า
$$bound = |M| + \sum_{l \in L} \min(|\{v \in V(G) : label(v) = l\}|, |\{v \in V(H) : label(v) = l\}|),$$

ค่า **bound** เกิดจากขนาดของ M ในปัจจุบันบวกกับผลรวมของ ค่าน้อยสุดของจำนวนปมที่มี **label** เดียวกันในกราฟ G และ H ทั้งหมด ยกตัวอย่างเช่นจากรูปก่อนหน้าในระดับการค้นที่สอง จะมีขนาด M เป็น 2 และมีคู่ปมที่มี **label** เหมือนกันคือ 2, 4, C กับ 5, D จะให้ค่าน้อยสุดของจำนวนปมเป็น 1 (ปมในกราฟ H) กับ 1 (เท่ากับคือ 1) ได้ค่า **bound** เป็น 4 เป็น **Upper bound** ของการค้นหาในครั้งนี้ (คิดในกรณีที่ว่าการเลือกครั้งต่อไปทั้งหมด จะมี **label** ที่เหมือนกันเหมือนเดิมจึงสามารถเลือกได้เป็นจำนวนมากสุด) ซึ่งหากค่า **bound** ที่เกิดจาก M ที่หาอยู่นั้นน้อยกว่าค่าขนาดของ M สูงสุดที่เคยได้จะไม่มีประโยชน์ในการทำต่อจึงทำการ **return** เพื่อ **backtrack** ไปทำในกรณีอื่นๆนั่นเอง

Code(c++)

```
#include<iostream>
#include<set>
#include<algorithm>
#include<map>
using namespace std;
set<pair<int,int>> maximumcomsub={};
map<int,set<int>> NG;//map with
key=vertex in G and key = adj to key
map<int,set<int>> NH;
set<int> VG;
set<int> VH;

int sigma(set<pair<set<int>,set<int>>>
future){//ฟังก์ชัน
    int c=0;
    for(auto x:future){
        if(x.first.size()>=x.second.size()){
            c+=x.second.size();
        }else{
            c+=x.first.size();
        }
    }
    return c;
}

int selectVertex(set<int> S){//ฟังก์ชัน
    int m=0;int k;
    for(auto x:S){
        if(NG[x].size() > m){
            m=NG[x].size();
            k=x;
        }
    }
    return k;
}

pair<set<int>,set<int>>
selectlabelclass(set<pair<set<int>,set<int>>>
future){
    int min=999999999;
    pair<set<int>,set<int>> R;
    for(auto x :future){
        int a =x.first.size();
        int b =x.second.size();
        int m=max(a,b);
        if(m<min){
            R=x;
            min=m;
        }
    }
    return R;}

set<int> adjacent(int v , char type){
    if(type == 'H'){
        return NH[v];
    }
    if(type == 'G'){
        return NG[v];
    }
    if(type == 'h'){
        set<int> S1;
        for(auto x:VH){
            if(NH[v].find(x)== NH[v].end() &&
x!=v){
                S1.insert(x);
            }
        }
        return S1;
    }
}

set<int> intersection(set<int> s1, set<int>
s2){
    set<int> ret;
    for(auto x:s1)for(auto y
:s2)if(x==y)ret.insert(x);
    return ret;
}

void MCIS(set<pair<set<int>,set<int>>>
future, set<pair<int,int>> M){
    if(M.size()==maximumcomsub.size())
    maximumcomsub=M;
    int bound =M.size()+sigma(future);
    if(bound<= maximumcomsub.size())return;
    pair<set<int>,set<int>> samelabel=
selectlabelclass(future);
    int v = selectVertex(samelabel.first);
    for(auto w:samelabel.second){
        set<pair<set<int>,set<int>>> future_1;
        set<int> G_1;//contain vertex in G with
label L
        set<int> H_1;//contain vertex in H with
label L
        set<int> G_2;//contain adjacent v
set<int> G_3;//contain not adjacent v
set<int> H_2;//contain adjacent w
set<int> H_3;//contain not adjacent w
        for(auto x : future){
            G_1 =x.first;
            H_1 =x.second;
            G_2=
intersection(G_1,adjacent(v,'G'));
            H_2=
intersection(H_1,adjacent(w,'H'));
            G_3= intersection(G_1,adjacent(v,'g'));
            H_3=
intersection(H_1,adjacent(w,'h'));
            if(G_2.size()>0 && H_2.size()>0){
                future_1.insert({G_2,H_2});
            }
            if(G_3.size()>0 && H_3.size()>0){
                future_1.insert({G_3,H_3});
            }
        }
        set<pair<int,int>> M_tmp=M;
        M_tmp.insert({v,w});
        //cout<<v<<" "<<w<<" ";
        MCIS(future_1,M_tmp);
    }

    set<int> g1={};
    for(auto x:samelabel.first){
        if(x!=v)g1.insert(x);
    }
    set<pair<set<int>,set<int>>>
futureUnmatch={};
    for(auto x:future){
        if(x!=samelabel)futureUnmatch.insert(x);
    }
    if(g1.size()!=0){
        futureUnmatch.insert({g1,samelabel.second
});
        MCIS(futureUnmatch,M);
    }

}

int main(){
    int k,nG,nH;cin>> k >> nG >> nH;

    set<pair<int,int>> M={};
    for(int i=0;i<nG;i++){
        int v; cin>> v;
        VG.insert(v);
    }
    for(int i=0;i<nH;i++){
        int v; cin>> v;
        VH.insert(v);
    }
    int edgeG,edgeH; cin >>edgeG>> edgeH;

    for(int i=0;i<edgeG;i++){//edge in graph G
        int a,b;cin >> a >> b;
        NG[a].insert(b);
        NG[b].insert(a);
    }
    for(int i=0;i<edgeH;i++){//edge in graph H
        int a,b;cin >> a >> b;
        NH[b].insert(a);
        NH[a].insert(b);
    }

    MCIS({{VG,VH}},M);

    if(maximumcomsub.size()>=k)cout<<"YES\n"
;
    else cout<<"NO\n";
    //for(auto
x:maximumcomsub)cout<<x.first<<" :"  
<<x.second<<"\n";
}
```

Line by line code explanation

```
1. void MCIS(set<pair<set<int>,set<int>>> future,
   set<pair<int,int>> M){
2. if(M.size()>maximumcomsub.size())
   maximumcomsub=M;
3. int bound =M.size()+sigma(future);
4. if(bound<= maximumcomsub.size())return;
5. pair<set<int>,set<int>> samelabel=
   selectlabelclass(future);
6. int v = selectVertex(samelabel.first);
7. for(auto w:samelabel.second){
8. set<pair<set<int>,set<int>>> future_1;
9. set<int> G_1;//contain vertex in G with label L
10. set<int> H_1;//contain vertex in H with label L
11. set<int> G_2;//contain adjacent v
12. set<int> G_3;//contain not adjacent v
13. set<int> H_2;//contain adjacent w
14. set<int> H_3;//contain not adjacent w
15. for(auto x : future){
16. G_1 =x.first;
17. H_1 =x.second;
18. G_2= intersection(G_1,adjacent(v,'G'));
19. H_2= intersection(H_1,adjacent(w,'H'));
20. G_3= intersection(G_1,adjacent(v,'g'));
21. H_3= intersection(H_1,adjacent(w,'h'));
22. if(G_2.size()>0 && H_2.size()>0){
23. future_1.insert({G_2,H_2});
24. }
25. if(G_3.size()>0 && H_3.size()>0){
26. future_1.insert({G_3,H_3});
27. }
28. }
29. set<pair<int,int>> M_tmp=M;
30. M_tmp.insert({v,w});
31. //cout<<v<<" "<<w<<" ";
32. MCIS(future_1,M_tmp);
33. }

34. set<int> g1={};
35. for(auto x:samelabel.first){
36. if(x!=v)g1.insert(x);
37. }
38. set<pair<set<int>,set<int>>>
   futureUnmatch={};
39. for(auto x:future){
40. if(x!=samelabel)futureUnmatch.insert(x);
41. }
42. if(g1.size()!=0){
43. futureUnmatch.insert({g1,samelabel.second});
44. MCIS(futureUnmatch,M);
45. }

46. }
```

คำอธิบายของตัวโค้ดโดยalgorithmหลักๆจะอยู่ใน

(line 1)function MCIS โดยจะรับ parameterเป็น

future ที่กล่าวไปข้างต้นและ Mเป็นเซตของคำตอบปัจจุบัน

(line 2)เป็นการแทนที่คำตอบที่แท้จริงหากขนาดของM นั้นมากกว่า
ขนาดของ maximumcomsub

(line 3-4)เป็นการimplement branch and bound และ
ทำการreturn หากเข้าเงื่อนไข ซึ่งทำหน้าที่คล้ายbase case ในการ
recursive

(line5) เลือกlabel ที่เหมือนกันจากในset ของ future

(line 6) เลือก V ด้วยฟังก์ชัน selectVertex จากเซตของ

future G

(line 7-33)เป็นการlabel ค่าต่อไปที่มีการเชื่อมและไม่เชื่อมจาก v
ไปยังปมที่เหลือในเซตG_1และจาก wในเซต H_1 และนำเซตที่มีการ
labelเหมือนกัน มาทำrecursiveซ้ำ นั่นเอง

(line8) ทำการสร้างfuture_1ขึ้นมาเพื่อเก็บ labelต่อไป ที่

เหมือนกัน หลักจากการเลือกvและw

(line9-14) ทำการdefine ตัวแปร

(line 15 -26)จะเป็นการสร้างเซตใหม่สองเซตจากlabel class

เดียวกันใน ทุกๆlabel class ใน futureโดยเซตแรก

(line 18-19)จะเป็นpair ของเซตของ ปมที่มีเส้นเชื่อมกับvใน

G_1 คู่กับปมที่มีเส้นเชื่อมกับw ใน H_1 โดยจะไม่รวม v และw ลง
ไป เซตที่สอง

(line20-21)จะเป็นpair ของเซตของ ปมที่ไม่มีเส้นเชื่อมกับvใน

G_1 คู่กับปมที่ไม่มีเส้นเชื่อมกับw ใน H_1 โดยจะไม่รวม v และw

(line22-27)ลงไปเก็บในเซตของfuture_1แต่จะไม่เก็บเมื่อlabel

ฝั่งใดฝั่งหนึ่งมีค่าเป็นเซตว่างหมายความว่าในการrecursiveครั้งต่อไป
จะไม่เกิดการจับคู่จึงไม่มีประโยชน์ที่จะใส่ลงไปนั่นเอง

(line29-32)เป็นการเพิ่มvและw ลงในเซตของการจับคู่ปม และทำ
การเรียกrecursive ด้วย labelใหม่ในfuture_1

(line34-45)เป็นกรณีที่กำหนดให้vที่เลือกมานั้นไม่คู่กับปมใดเลยใน
เซตของH(unmatch)เพื่อในเคสที่ว่าหากไม่ได้จับคู่vที่เลือกมานั้นอาจจะ
ให้ผลลัพธ์ที่ดีกว่าในการรันครั้งต่อไปได้ โดยนำv ออกจากเซตGและสร้าง
futureใหม่ซึ่งนำเอาlabelของvตัวนั้นออกและทำการเรียก
recursiveต่อไป

Time complexity

ในการค้นของแต่ละnode จะมีtime complexity เป็น $O((g + h)^2)$ แต่ต้องทำหลายครั้งและจำนวนครั้งไม่แน่นอน จึงใช้ความรู้ที่ว่า การหา MCIS นั้นสามารถลดรูปมาจาก clique problem ซึ่งมี time complexity เป็น $O(2^n)$ จึงสรุปว่า MCIS มี time complexity เป็น $O(2^n)$ นั่นเอง

จากโปรแกรมข้างต้นจะมีการรับ Input ดังนี้

บรรทัดแรกรับ K, nG, nH คือจำนวนของ MCIS ที่ต้องการ จำนวนปมในกราฟ G และจำนวนปมในกราฟ H ตามลำดับ โดย $K \geq 1, nG \geq 0, nH \geq 0$

สองบรรทัดต่อไปจะรับชื่อปมตามจำนวนปม nG และ nH (ห้ามมีชื่อปมซ้ำ)

บรรทัดที่ 4 รับจำนวน nVg, nVh ซึ่งคือจำนวนเส้นเชื่อมในกราฟ G และ H ตามลำดับ โดย $nVg, nVh \geq 0$

บรรทัดที่เหลืออีก nVg จะรับเส้นเชื่อมระหว่างแต่ละปมในกราฟ G และ อีก nVh บรรทัดจะรับเส้นเชื่อมในกราฟ H

Output เป็นคำตอบ YES หรือ NO ว่ากราฟทั้งสองที่ให้ มี isomorphic subgraph ขนาดไม่น้อยกว่า k อยู่หรือไม่

Interesting case

1. กราฟตัวอย่าง
2. กราฟที่ไม่มีเส้นเชื่อมเลยและกราฟที่มีเส้นเชื่อมจำนวนมาก
3. เส้นตรงและจุด
4. วงวนวงใหญ่และวงวนวงเล็ก
5. กราฟใดๆ และ complete graph ที่มีจำนวน vertex เท่ากัน

*สามารถนำค่าไปใส่ในโปรแกรมด้านบนได้เลย

1.

4 6 4

1 2 3 4 5

6

7 8 9 10

8 4

1 2

1 3

1 4

1 5

2 4

3 5

4 6

5 6

7 8

7 9

7 10

8 10

2.

2 5 5

1 2 3 4 5

6 7 8 9 10

7 0

1 2

1 5

2 3

2 4

2 5

3 4

3 5

3.

1 3 1

1 2 3

4

0 2

1 2

2 3

4.

3 8 4

1 2 3 4 5

6 7 8

9 10 11

1 2

8 4

1 2

1 8

2 3

3 4

4 5

5 6

6 7

7 8

9 10

9 12

10 11

10 11

11 12

5.

4 6 6

1 2 3 4 5

6

7 8 9 10

11 12

9 15

1 3

1 6

2 3

3 4

3 5

3 6

4 5

4 6

5 6

7 8

7 9

7 10

7 11

7 12

8 9

8 10

8 11

8 12

9 10

9 11

9 12

10 11

10 12

11 12

Video

<https://www.youtube.com/watch?v=qBS9UJSnz94>

Ref

https://en.wikipedia.org/wiki/Graph_isomorphism

<https://www.ijcai.org/Proceedings/2017/0099.pdf>

<https://stackoverflow.com/questions/14643948/how-can-i-find-maximum-common-subgraph-of-two-graphs>