DS Final Project Report

一、K-core

<參考影片>：https://www.youtube.com/watch?v=rHVrgbc_3JA

關於 K-core 的部分，我參考了助教提供的 ppt 以及網路上的相關影片，從 ppt 以及影片的步驟，我發現找 K-core 的關鍵是由 degree 小的 vertex 下手。**由 degree 小的 vertex 開始移除到 degree 大的 vertex。**在移除的過程中，假設今天我要移除 A 點，那麼下一步就是將有跟 A 點相連的點的 degree 全部減一，然後移除 A 點的當下，他自己的 degree 就是他的 coreness。重複做完以上的動作，直到所有的點都被拔光。如此一來，就能找出所有點的 coreness。

以下解釋我是如何操作 K-core：

```
while(cin >> v1 >> v2){
    if(v1 == v2)
        continue;
    vertex[v1].push_back(v2);
    vertex[v2].push_back(v1);
    vertex_kb[v1].insert(v2);
    vertex_kb[v2].insert(v1);
}

set<pair<int, int> > vert;//<deg, vertex_id>
set<pair<int, int> >::iterator it1;

for(int i = 0; i < V ; i++){
    vert.insert({vertex[i].size(), i});
    deg.push_back(vertex[i].size());
    coreness.push_back(vertex[i].size());
}
```

首先我用 vector<vector<int> > vertex 存取各點有相鄰的點(類似 adj list)，接著我開一個 set<pair<int, int> > vert，第一個位置存該點的 degree；第二點存該點的 id，其好處是 insert 進去 set 裡面的點，**會按照 degree 由小到大排好**。

接著記錄下該點的 degree，然後 insert 到 set 裡。

```
it1 = vert.begin();

while(!vert.empty())
{
    int vertex_id = it1->second;
    int vertex_deg = it1->first;

    coreness[vertex_id] = vertex_deg;
    vert.erase(it1);

    P[--cnt] = vertex_id;
    if(coreness[vertex_id] >= 499) //to count vertices' coreness >= 499 to find 500 >> clique
        kb_v++;

    for(int i = 0; i < vertex[vertex_id].size(); i++){
            if(deg[vertex[vertex_id][i]] > vertex_deg){
                vert.erase({deg[vertex[vertex_id][i]], vertex[vertex_id][i]});
                vert.insert({--deg[vertex[vertex_id][i]], vertex[vertex_id][i]});
            }
            else{
            }
    }
    it1 = vert.begin();
}
```

接下來由 degree 小的 vertex 開始移除，在移除前要先記錄其 degree 以及 id，移除的點的 coreness 就是其 degree。接著跑迴圈依次將有跟剛移除的點相鄰的點 degree 全部減一。**<注意>此迴圈不能將 degree 相同的點減一，因為他們同**

屬相同的 coreness。(K-core)

二、Clique

<參考影片>：https://www.youtube.com/watch?v=132XR-RLNoY&t=988s
<參考資料>：https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm
關於 Clique 的部份，我參考了以上的影片以及資料，發現有一個系統性找 Clique 的演算法：**Bron–Kerbosch algorithm**(BK algorithm)。以下為 BK algorithm 的 pseudocode：

```
algorithm BronKerbosch1(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    for each vertex v in P do
        BronKerbosch1(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

這個演算法主要有三個集合：R, P, X，R 是用來存 clique 的集合；P 是存可以跟 R 裡所有點相鄰的集合；X 是避免重複計算而存在的集合(目前還沒想到控制 X 的條件機制，所以暫且不管)。這個演算法就是不斷的將 P 的一個元素丟到 R，然後 P 再移除掉沒有跟 R 裡所有點相連的點，不斷重複這個動作，直到 P 這個級何為空集合，再 output 最大的 clique。

這個演算法可以算得更快，算得更快的條件在於 P 集合所挑選的點，如果該點所連到的點有很多，那麼就有很大的機會，在一開始就會有最大的 clique。這個就跟我們在前面算的 coreness 有很大的關係。(coreness 越大代表越有可能形成大的 clique)因此，**我所採用的作法是將 P 集合裡的點，由 coreness 大到 coreness 小依次排序**。
以下解釋我是如何操作 Clique：

```
int cnt = V;
int kb_v = 0;
itl = vert.begin();

while(!vert.empty())
{
    int vertex_id = itl->second;
    int vertex_deg = itl->first;

    coreness[vertex_id] = vertex_deg;
    vert.erase(itl);

    P[--cnt] = vertex_id;
    if(coreness[vertex_id] >= 499) //to count vertices' coreness >= 499 to find 5
        kb_v++;

    for(int i = 0; i < vertex[vertex_id].size(); i++){
            if(deg[vertex[vertex_id][i]] > vertex_deg){
                vert.erase({deg[vertex[vertex_id][i]], vertex[vertex_id][i]});
                vert.insert({--deg[vertex[vertex_id][i]], vertex[vertex_id][i]});
            }
            else{
            }
    }
```

vector<int> P 從後面依次存入由小到大 coreness 的 vertex id，所以 **P 裡面的 vertex coreness 順序是由大到小**

```
void BronKerbosch(set<int> R, vector<int> P){
    if(P.size() == 0){
        if(R.size() > MAX_clique.size()){
            MAX_clique.clear();
            Print_Clique(R);
            for(set<int>::iterator it = R.begin(); it != R.end(); ++it)
                MAX_clique.insert(*it);
            return;
        }
        else
            return;
    }
    else{
        bool put_to_R;
        vector<int> NEW_P;
        vector<int>::iterator it = P.begin();
        while(it != P.end()){
            R.insert(*it);
            int temp = *it;
            P.erase(it);
            for(vector<int>::iterator it2 = P.begin(); it2 != P.end(); ++it2){
                if(vertex_kb[temp].count(*it2) > 0)
                    put_to_R = 1;
                else
                    put_to_R = 0;
                if(put_to_R)
                    NEW_P.push_back(*it2);
            }
            BronKerbosch(R, NEW_P);
            R.erase(temp);

            NEW_P.clear();
            it = P.begin();
        }
    }
```

先看到 else 的部分(遞迴不斷呼叫的部分)，首先，P 每次都會將最前面的點移除掉並 insert 到 R 的集合，接著會有 NEW_P 這個 vector 去存 P 有跟剛放到 R 集合的點相鄰的點，接著再呼叫一次這個遞迴方程式，接著要把 R 以及 NEW_P 清空，iterater 繼續指到頭。(**把 R 清空是因為原本 insert 的那個點可以不選**；把 NEW_P 清空是因為 NEW_P 是要暫存根移除點相鄰的點，結束要清空，不然會影響到下回合)

接著看到 if 的部分，遞迴終止條件為 P 是空的，如果 P 是空的，且 R 的 size 比現階段的 clique 還大，那就更新 MAX_clique。

在 Clique 這邊有遇到許多困難，第一個是 BK 演算法的 X 集合，這個集合應該是加速的關鍵，因為可以避免做到許多重複的 case，研究了很久還是不太了解他的原理，寫不太出來。

還有**如果 P 沒有按照 coreness 排序**，光要印到 clique = 500 就有很大的難度，因為老實說 BK 演算法雖然是有系統地找出 clique，但是因為 vertex 還是太多了，這樣互相檢查要花上很多時間，**由此證明第一個挑選 P 的 pivot 是極為重要的**。這次的 final project 真的很挑戰，花上了很多時間看資料以及跟同學討論才打出來，謝謝老師以及助教們，你們辛苦了。