Algorithm PA3 Report

b06502152, 許書銓

Data Structures

在這次作業裡，我大致將題目會用到的資料分成以下架構a. Edge, b. Vertice, c. Graph, d. Disjointset, e. rmEdges，並且在這樣的架構下，寫成class來操作，如以下描述，

a. Edge

```
class edge{
public:
    int u;
    int v;
    int w;
    edge(int u, int v, int w);
};
```

資料結構：自定義class edge

說明：(i). u屬性表示edge的開頭，以int儲存。

(ii). v屬性表示edge的結尾，以int儲存。

(iii). w屬性表示edge的weight，以int儲存。

b. Vertice

```
class Vertice{
public:
    int parent;
    int rank;
    int color; // 0 for white, 1 for gray, 2 for black
    Vertice();
};
```

資料結構：自定義class Vertice

說明：(i). parent屬性表示其predecessor，以int儲存。

(ii). rank屬性表示其在做disjointset時的rank大小，以int儲存。

(iii). color屬性表示其在做DFS時被visited的情形，以int儲存。

c. Graph

```
class Graph{
public:
    string type;                                    //feature for undirected or directed graph
    int num_v;                                      //feature for num of vertices
    int num_e;                                      //feature for num of edges
    int has_cycle;                                  //record has cycle or not, if y/1, else n/0
    vector<vector<edge> > adj_list;                 //using adjency list to store edges
    vector<edge> E;                                 //using vetor to store edges

    Graph();
    Graph(string, int, int);
    void removeAll(vector<edge>);                   //removing edges in adj_list and E, after running reversed_Kruskal()
    bool cycleDetect(edge& );                       //decting if add this edge will lead to cycle or not
    bool cycleDetect_helper(int, int, Vertice* );   //running DFS, source on vertice u, checking will it lead back to source
    void printEdges();                              //printing all edges in E
    void printAdjList();                            //printing all edges in adj_list
    void addEdge(edge);                             //adding edges while constructing graph
    void removeEdge(edge);                          //removing edge e in E and adj_list
    bool hasCycle();                                //check the graph having cycle or not, if y/1, else n/0
    void DFS();                                     //DFS for all graph
    void DFS_Visit(Vertice *, int);                 //DFS_visit for all graph
    void sortEdges();                               //using Merge Sort in PA1 to sort all edges in E
    void MergeSort(vector<edge>&);
    void MergeSortSubVector(vector<edge>&, int, int);
    void Merge(vector<edge>&, int, int, int, int);
};
```

資料結構：自定義class Graph

說明：(i). type屬性表示其圖為directed graph或是undirected graph，u表示undirected，d為directed，以string儲存。

(ii). num_v 表示其vertice數量，以int儲存。

(iii). num_e 表示其edge數量，以int儲存。

(iv). has_cycle表示graph裡存在cycle，1/exist, 0/not exist，以int儲存。

(v). adj_list為adjacency list，以vector<vector<edge> >儲存。

(vi). E為edge的集合，儲存所有edge，以vector<edge>儲存。

d. Disjointset

```
class disjointSet{
public:
    Vertice *set;
    void makeSet(int num_v);
    void Union(int u, int v);
    int findSet(int u);
    void Link(int u, int v);
    disjointSet(int num_v);
};
```

資料結構：自定義class disjointSet

說明：set表示disjointset裡面目前vertice的狀況，以動態配置的array儲存。

e. rmEdges

```
    vector<edge> rmEdges;
```

資料結構：vector<edge>

說明：以vector儲存最後決定要刪除的edge。

Findings

a. Fin/fstream

實際上在使用fin的時候，發現一次擷取一行，在擷取中間的內容，會比一次讀一個資料快速讀完，我想在資料量大一點的時候會有比較顯著影響，看到這次最大test case edge數量到50000000就決定這樣改。

b. DFS/Cycle_detect

在directed graph的題目裡，因為過程中會檢查加入這個edge會不會形成cycle，故比較好的做法是直接從edge的v端(end 端點)點開始跑，看看會不會有機會跑到edge的u端點(start端點)，會比從graph跑一次完整的DFS檢查graph裡面有沒有cycle來得有效率。

c. Sorting

在directed graph題目裡，因為我的做法是先跑kruskal，跑完後會得到要刪除的edges，再由大到小依序加回去graph裡，檢查有無cycle，沒有cycle就順利加回去。然而最後在實作的時候發現，不同的sorting algorithm除了時間上有差異，在對於處理euqal key的表現也不樣，後來發現與vertor.sort相比，如果使用merge sort來排序weight大小，會得到較小的cycle weight，然而時間上較長。

但上述討論僅於public_case裡面有merge sort比較好的結果，很有可能換了輸入進來，得到的結果不一定會較好，故後來我選擇以時間處理上較快速的vector sort來處理sorting問題。