B12901066 電機二 吳孟峰 Programming Assignment #3 Report

1. Data structure

- **How do I store graphs**

One major issue when doing this problem is that **the edge will change overtime**, meaning that we have to frequently modify the graph. Therefore, I use both **Adjacency matrix** and **Adjacency list**. While adjacency list stores original graph's edges, adjacency matrix double checks if the edge exists. That is

$$\text{Chords}[i][j] = \begin{cases} 1, \text{chord } ij \in \text{C}, \text{and it still exists} \\ 0, \text{chord } ij \notin \text{C} \\ -1, \text{chord } ij \in \text{C}, \text{and it's removed} \end{cases}$$

- **Other data structure used**

I design a binary (min)heap class to implement Prim's method of obtaining MST, consisting of follow members:

```
class binary_heap {
    public:
        binary_heap();
        ~binary_heap();
        vertex* extract_min();
        void decrease_key(int index, int key);
        void insert(vertex* vertex);
        int get_size() {return size;}

    private:
        vertex* heap[10000] = {};
        int size = 0;
        void min_heapify(int index);
        void build_min_heap();
        void swim(int index);
};
```

The "vertex" is a struct that stores some property of vertices that is useful when implementing graph algorithms:

```
struct vertex {
    int index;
    int d = 0; // degree
    vertex* p = NULL;
    colors color = WHITE;
    int nowCheck = 0;
    // candy variable for saving the status of checking edges

    int key = INF;
    bool used = false;
    int heapInd = 0;
};
```

2. Algorithm explanation and Time complexity analysis

- Undirected Unweighted Graph

Notice that a connected undirected graph $G = (V, E)$ has no cycle if and only if it has $|V| - 1$ edges. Therefore, **the minimal cost to break a cycle is always $|\boldsymbol{E}| - |\boldsymbol{V}| + \mathbf{1}$**. Still, we have to make sure that the new graph is weakly connected, therefore we **use (undirected version) DFS to form a depth-first tree, and remove all edge that is not in the tree.** The time complexity is $\boldsymbol{\Theta(V + E)}$.

- Undirected Weight Graph

The conclusion remains unchanged. That is, we can (and must) only left $|V| - 1$ edges in the graph. Thus, all left edge in the graph should be the most expensive edges. Therefore, these edges form a maximum spanning tree. Thus, to brake cycles in the graph, we use Prim's algorithm to find the maximum spanning tree[#], and remove all edges that is not in the maximum spanning tree. The time complexity is $\boldsymbol{O(E \lg V)}$.

To do so, we can negate all weight and apply Prim's algorithm

- Directed Weight Graph

If we treat the graph as an undirected graph and form a maximum spanning tree, then the tree is, obviously, acyclic. But the result is way far than good. To improve our solution, we try to add an edge in the solution set (denoted e) back to the graph, and use DFS to check if that causes any cycle, we markdown the lightest edge. Eventually we obtain a set of edges S that need to be removed, we compare which one is better, "return edge e" or "remove all edges in S". If we choose to "remove all edges in S", all edges in S are added to the solution set. We repeat this process until no edge can be re-add to the graph. The overall time complexity depends on the number of iterations of optimization step, if we denote it as C, then the complexity can be express as $O(E \lg V) + O\big(C(V + E)\big) = \boldsymbol{O\big(E \lg V + C(V + E)\big)}$.