演算法導論 Lab2_report

B101016 曹寓恆

策略:

這次作業因為有 exist routing，因此 kruskal 相較於 prims 更適合 Lab2。

資料結構:

```
struct pin
{
    int x, y;
    pin(int _id, int _x, int _y) : x(_x), y(_y) {}
};
```

```
struct edge
{
    int p1, p2; // connected points' index
    int d;      // |x1 - x2| + |y1 - y2|
    edge(int _p1, int _p2) : p1(_p1), p2(_p2), d(dist(_p1, _p2)) {}
    bool operator<(const edge &e) const { return d < e.d; }
};
```

```
struct disjoint_set
{
    vector<int> ds;
    disjoint_set(int numV)
    {
        for (int i = 0; i < numV; i++)
            ds.push_back(i);
    }
    int find(int x) { return x == ds[x] ? x : (ds[x] = find(ds[x])); }
    void merge(int x, int y) { ds[find(y)] = find(x); }
};
```

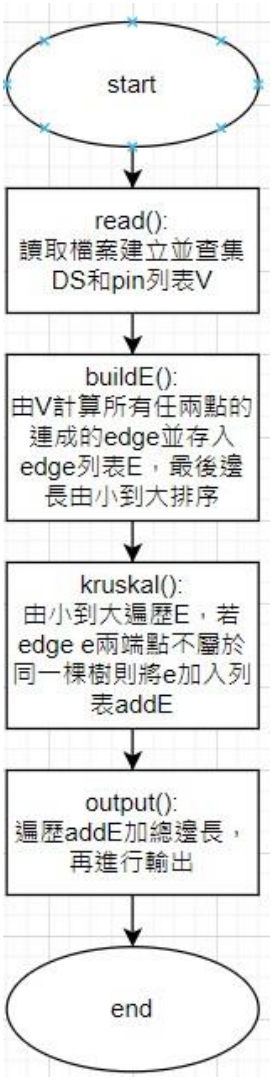
主程式架構:

```
int main(int argc, char **argv)
{
    string inF = argc == 3 ? argv[1] : "input.txt";
    string outF = argc == 3 ? argv[2] : "output.txt";

    disjoint_set DS = read(inF);
    buildE();
    vector<edge *> addE = kruskal(DS);
    output(outF, addE);

    return 0;
}
```

流程圖 & 時間複雜度:



1.read():

列表 V 大小取決於 number of pins，因此時間複雜度 = O(|V|)，尋找和更新併查集的時間複雜度都是 O(α(|V|))。整體時間複雜度為 O(|V| + |E’ | * α(|V|))，

|E’ | 表示 exist routing 的個數，又因 exist routing 不存在 cycle 故|E'| < |V| - 1 = O(|V|)，故整體時間複雜度為 O(|V| * α(|V|))。

2. buildE():

因為是無向圖，任兩點連線的組合數為|E| = 1+2+3+ … + |V| = (1+|V|)(|V|) / 2，因此時間複雜度為 O(|E|) = O(|V|^2)。

3. kruskal():

首先要對列表 E 遞增排序時間複雜度為 O(|E| * lg|E|)，

接著會依序取出並檢查 edge e 兩端點是不是屬於同一棵樹需要分別執行一次尋找併查集，若 e 的兩端點屬於不同的樹，則先更新併查

集，再將 e 加入存放 spanning edge 的列表 addE，重複這個步驟直到完成的

時間複雜度為 O(|E| * α(|V|)) 。整體時間複雜度為 O(|E| * lg|E|)。

4. output():

形成 MST 最多會往 addE 加入|V| - 1 條 edge，故遍歷
addE 計算 cost 的時間複雜度為 O(|V|)，接下來再遍歷
一次 addE 輸出新增 edge 的兩個端點也是 O(|V|)，整體
時間複雜度為 O(|V|)。

5. main():

綜上所述，整體時間複雜度為 O(|V| * α(|V|)) + O(|V|^2) + O(|E| * lg|E|) + O(|V|) = O(|E| * lg|E|) = O(|E| * lg|V|)。