F74109017_hw14 葉惟欣

```c
//f74109016_葉惟欣 finished in 2021/11/1
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
//height union
typedef struct node NODE;
struct node{
    int parent;
    int height;
};
void create1(NODE *array1,int element){
    int i;
    for(i=0;i<element;i++){
        (array1+i)->parent=-1;
        (array1+i)->height=1;
    }
}
void create(int *array,int element){
    int i;
    for(i=0;i<element;i++){
        *(array+i)=-1;
    }
}
int find_root1(NODE *array1,int tree1){
    while((array1+tree1)->parent>-1){
        tree1 = (array1+tree1)->parent;
    }
    return tree1;
```

```
}
int find_root(int *array,int tree1){
    while(*(array+tree1)>-1){
        tree1 = *(array+tree1);
    }
    return tree1;
}
int find_height(NODE *array1,int root,int element){
    int i,height=1,max =0;
    for(i=0;i<element;i++){
        if((array1+i)->parent==root){
            int height_temp = (array1+i)->height;
            if(height_temp>max){
                max=height_temp;
            }
        }
    }
    return max+1;
}

void same1(NODE *array1,int tree1,int tree2){
    int root_tree1 = find_root1(array1,tree1);
    int root_tree2 = find_root1(array1,tree2);
    if(root_tree1==root_tree2){
        printf("true\n");
    }
    else{
```

```
2
10 5
height_union 0 1
-1 0 -1 -1 -1 -1 -1 -1 -1 -1
height_union 2 0
-1 0 0 -1 -1 -1 -1 -1 -1 -1
find 2
0
-1 0 0 -1 -1 -1 -1 -1 -1 -1
same 2 1
true
-1 0 0 -1 -1 -1 -1 -1 -1 -1
same 2 4
false
-1 0 0 -1 -1 -1 -1 -1 -1 -1
```

```
    }
    else{
        printf("false\n");
    }
}
void height_union(NODE *array1,int tree1,int tree2,int element){
    int root_tree1 = find_root1(array1,tree1);
    int root_tree2 = find_root1(array1,tree2);
    if(root_tree1!=root_tree2){
        if((array1+root_tree1)->height < (array1+root_tree2)->height){
            (array1+root_tree1)->parent = root_tree2;
        }
        else if((array1+root_tree1)->height > (array1+root_tree2)->height){
            (array1+root_tree2)->parent = root_tree1;
        }
        else{
            (array1+root_tree2)->parent = root_tree1;
            (array1+root_tree1)->height+=1;
        }
    }
}
void same(int *array,int tree1,int tree2){
    int root_tree1 = find_root(array,tree1);
    int root_tree2 = find_root(array,tree2);
    if(root_tree1==root_tree2){
        printf("true\n");
    }
    else{
        printf("false\n");
```

```
        printf("false\n");
    }
}
//weight union
void weight_union(int *array,int tree1,int tree2){
    /*union the sets with roots i and j, i!=j,using the weighting rule. par
    int root_tree1 = find_root(array,tree1);
    int root_tree2 = find_root(array,tree2);
    if(root_tree1!=root_tree2){
        int temp = *(array+root_tree1) + *(array+root_tree2);

        if(*(array+root_tree1) > *(array+root_tree2)){
            *(array+root_tree1) = root_tree2;
            *(array+root_tree2) =temp;
        }
        else{
            *(array+root_tree2) = root_tree1;
            *(array+root_tree1) = temp;
        }
    }
}
int collapsingFind1(NODE * array1,int tree1,int element){
    int root = find_root1(array1,tree1);
    while((array1+tree1)->parent>-1){
        int temp = (array1+tree1)->parent;
        (array1+tree1)->parent = root;
        (array1+tree1)->height = 1;
        tree1 = temp;
```

```
109         (array1+tree1)->height = 1;
110         tree1 = temp;
111     }
112     (array1+tree1)->height = find_height(array1,tree1,element);
113     (array1+root)->height = find_height(array1,root,element);
114     return root;
115 }
116 int collapsingFind(int * array,int i){
117     //find the root of the tree containing element i. Use the collapsing rule t
118     int root,trail,lead;
119     for(root = i;*(array+root)>=0;root = *(array+root)){
120
121     }
122     for(trail = i; trail != root;trail = lead){
123         lead = *(array+trail);
124         *(array+trail) = root;
125     }
126     return root;
127 }
128 int main(){
129     int round,i,j,k,element,operation_time;
130     char string[5];
131     int *ptr ;
132     scanf("%d",&round);
133     clock_t start1 = clock();
134     for(i=0;i<round;i++){
135         scanf("%d %d",&element,&operation_time);
136         int array[element];
137         NODE array1[element];
```

```
136         int array[element];
137         NODE array1[element];
138         int mode=0;
139         create(array,element);
140         for(j=0;j<operation_time;j++){
141             scanf("%s",string);
142             if(strcmp(string,"height_union")==0){
143                 int tree1,tree2;
144                 mode = 0;
145                 if(j==0){
146                     create1(array1,element);
147                 }
148                 scanf("%d %d",&tree1,&tree2);
149                 height_union(array1,tree1,tree2,element);
150             }
151             else if(strcmp(string,"weight_union")==0){
152                 int tree1,tree2;
153                 mode = 1;
154                 if(j==0){
155                     create(array,element);
156                 }
157                 scanf("%d %d",&tree1,&tree2);
158                 weight_union(array,tree1,tree2);
159             }
160             else if(strcmp(string,"find")==0){
161                 int goal;
162                 scanf("%d",&goal);
163                 if(mode==0){
164                     printf("%d\n",collapsingFind1(array1,goal,element));
```

```
164                     printf("%d\n",collapsingFind1(array1,goal,element));
165                 }
166                 else{
167                     printf("%d\n",collapsingFind(array,goal));
168                 }
169             }
170             else if(strcmp(string,"same")==0){ //在同一個集合
171                 int tree1,tree2;
172                 scanf("%d %d",&tree1,&tree2);
173                 if(mode==0){
174                     same1(array1,tree1,tree2);
175                 }
176                 else{
177                     same(array,tree1,tree2);
178                 }
179             }
180         }
181     }
182     clock_t end = clock();
183     double elapsedTime1 = (double) (clock() -start1)/CLOCKS_PER_SEC;
184     printf("weight_union elapsedTime : %.20f\n",elapsedTime1);
185     return 0;
186 }
```

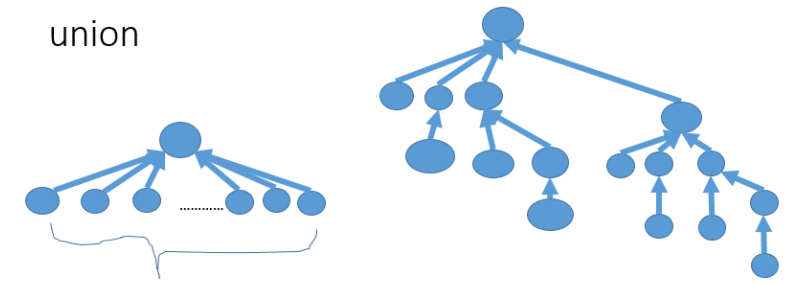
因為 height_union 的 array 紀錄的是 parent。所以聯集的時候要先找到要聯集的節點的 root➔find_root()。如果 root 相同就不聯集。高度較小的就接在高度較高的 root 下面。

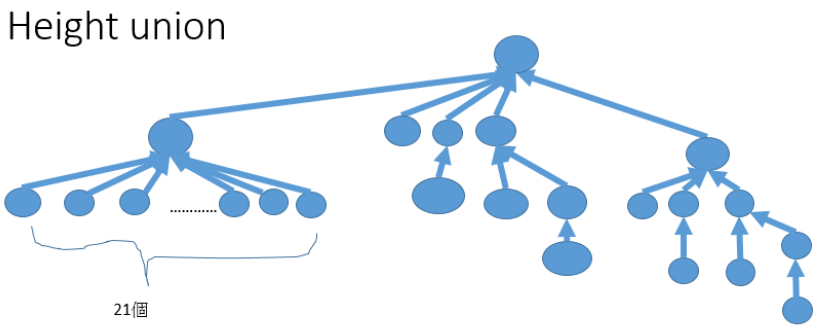
```
2
10 5
weight_union 0 1
-2 0 -1 -1 -1 -1 -1 -1 -1 -1
weight_union 2 0
-3 0 0 -1 -1 -1 -1 -1 -1 -1
find 2
0
-3 0 0 -1 -1 -1 -1 -1 -1 -1
same 2 1
true
-3 0 0 -1 -1 -1 -1 -1 -1 -1
same 2 4
false
-3 0 0 -1 -1 -1 -1 -1 -1 -1
```

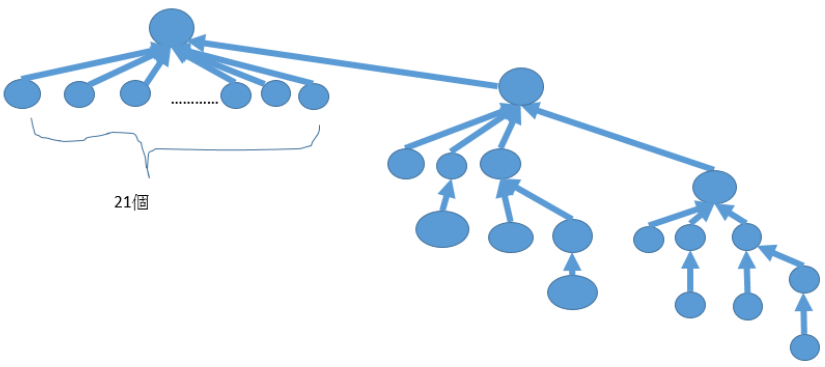
weight_union 函數，array 裡面紀錄的是 parent，但是 root 紀錄的節點大小。

Weight_union 的 array，除了 root 外其餘都是記錄他的 parent。而 root 紀錄的是目前有多少個節點在那個集合裡面，還要取負號。因為原先每個 array 的元素自己就是一個 union 所以 array 裡面存的都是-1，聯集後就直接相加。所以每當要聯集的時候就先去找該節點的 root 找到後才能比數量判斷誰要當子樹誰要當節點。

Collapsing find()的操作

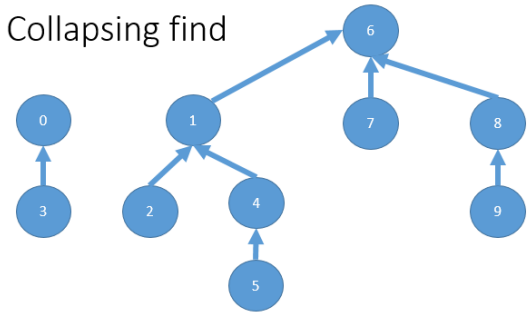


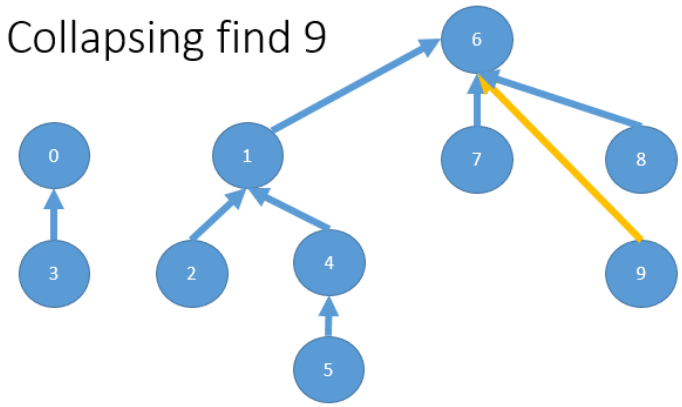




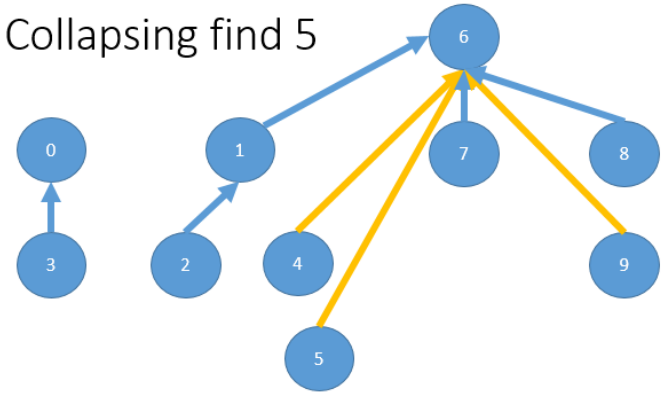
可以看到 height_union 聯集後的高度為 5。
Weight_union 聯集後的高度為 6。

Collapasing find 的操作如下舉例



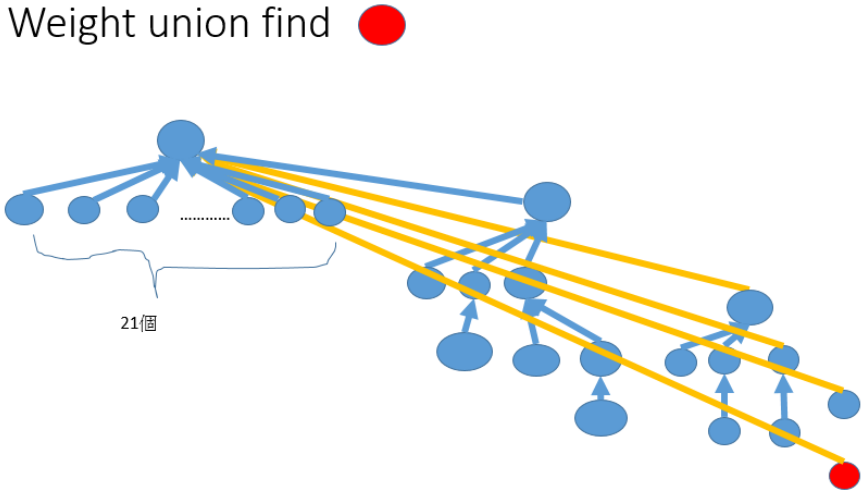


將從 root 到 9 經過的全都直接連到 6➔collasping find 是一種高度壓縮。

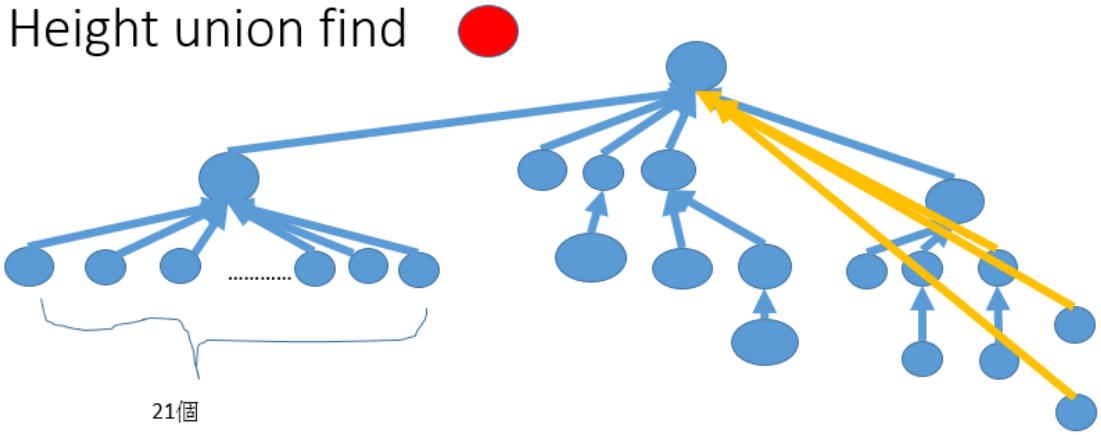


將從 root 到 9 經過的全都直接連到 6(經過 4 跟 5)➔collasping find 是一種高度壓縮。

從這些圖可以看出在 find 的時候做壓縮，因此我們得知，如果 find 是 leaf 節點的話，collasping find 耗費的時間與樹的高度有關。
再從剛剛 weight_union 做的結果來看。要 collasing find 的數量為 4



從剛剛 weight_union 做的結果來看。要 collasing find 的數量為 3

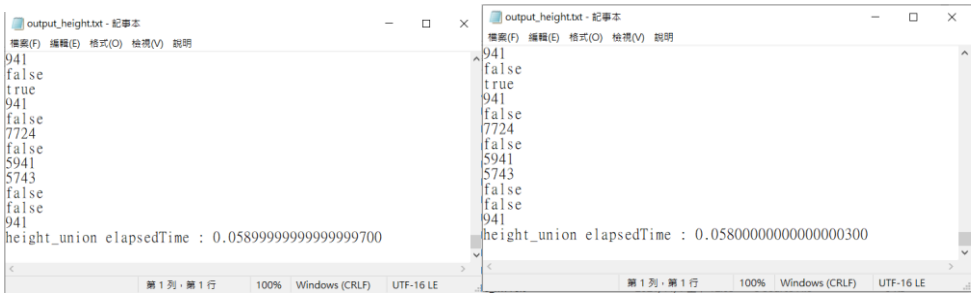


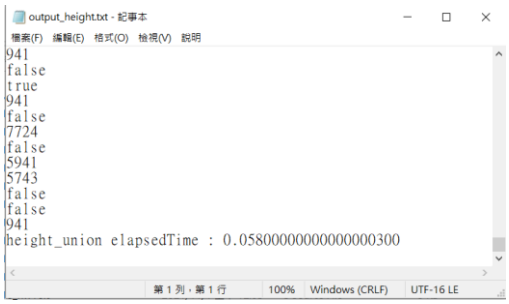
因為 height union 的時候有在注重減少 tree 的高度，而 weight union 則是注重樹的節點數量。而 collasing find 跟高度有關係。所以會在 height union 的時候效率比較高。

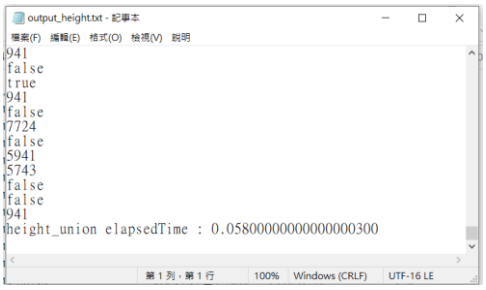
實驗:用 homework13 的 input_02

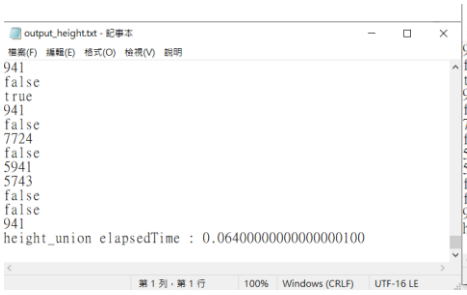
將程式碼的 weight_union 或 height_union 先改成 union 即可。

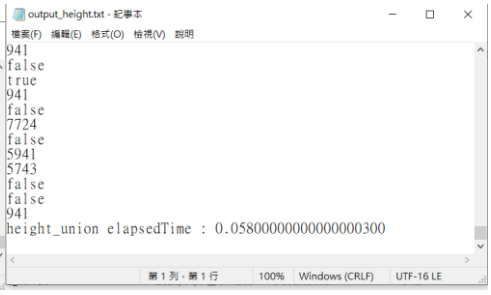
Height_union:

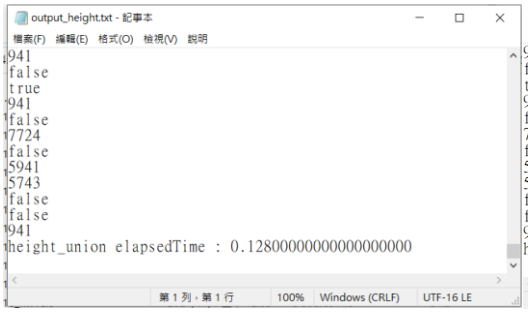


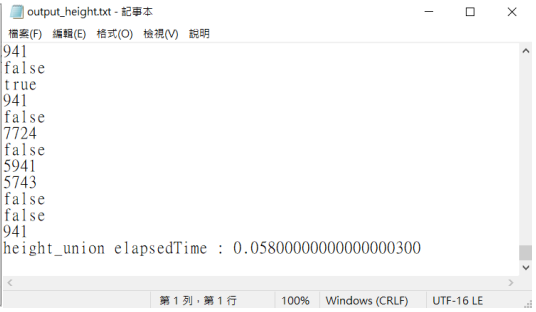


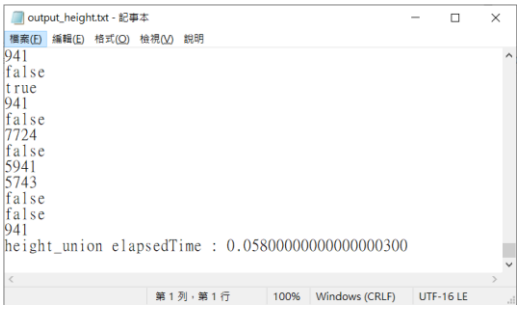




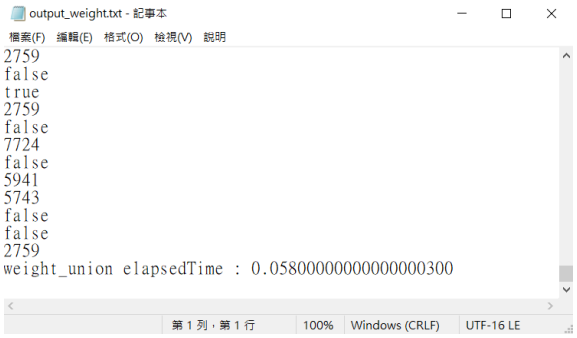


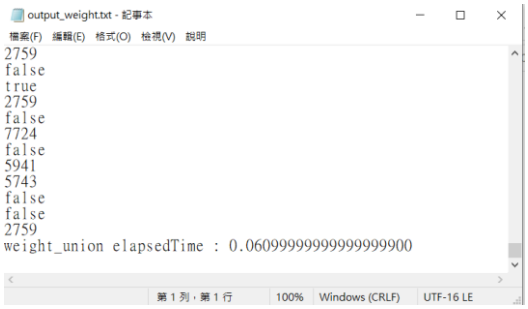


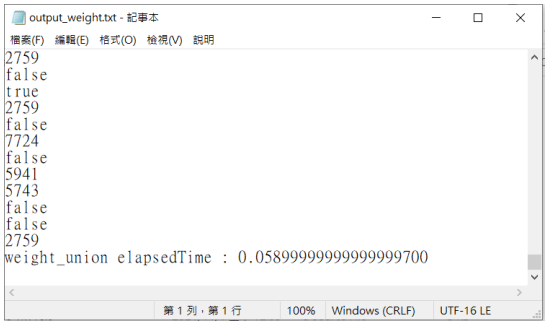


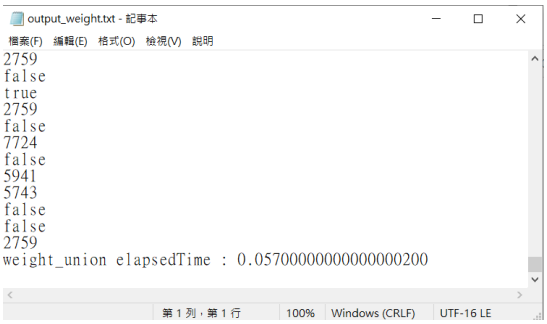


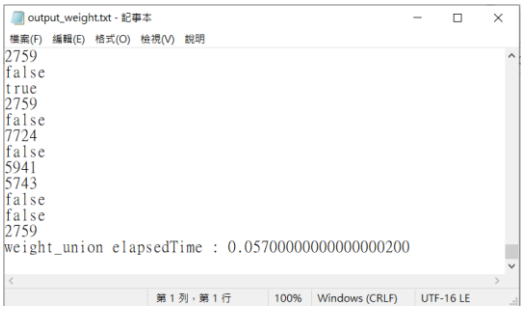
Weight_union:

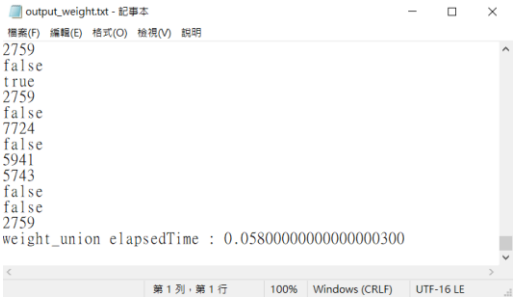


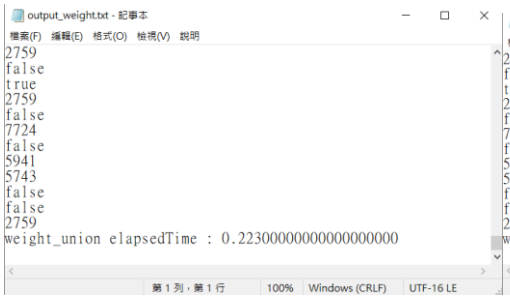


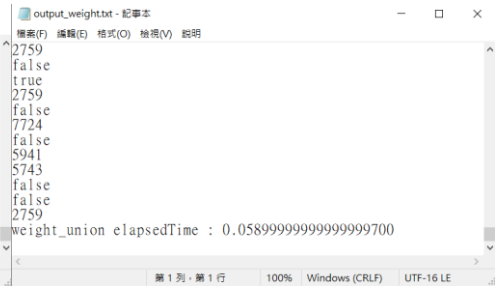

output_weight.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
2759
false
true
2759
false
7724
false
5941
5743
false
false
2759
weight_union elapsedTime : 0.05899999999999999700
第 1 列，第 1 行 100% Windows (CRLF) UTF-16 LE

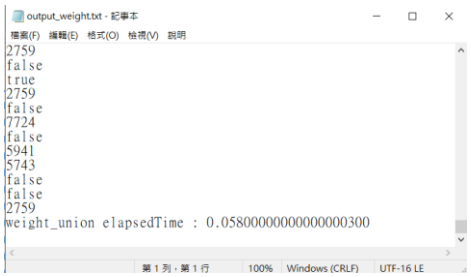

output_weight.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
2759
false
true
2759
false
7724
false
5941
5743
false
false
2759
weight_union elapsedTime : 0.05700000000000000200
第 1 列，第 1 行 100% Windows (CRLF) UTF-16 LE


output_weight.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
2759
false
true
2759
false
7724
false
5941
5743
false
false
2759
weight_union elapsedTime : 0.05700000000000000200
第 1 列，第 1 行 100% Windows (CRLF) UTF-16 LE


output_weight.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
2759
false
true
2759
false
7724
false
5941
5743
false
false
2759
weight_union elapsedTime : 0.05800000000000000300
第 1 列，第 1 行 100% Windows (CRLF) UTF-16 LE


output_weight.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
2759
false
true
2759
false
7724
false
5941
5743
false
false
2759
weight_union elapsedTime : 0.22300000000000000000
第 1 列，第 1 行 100% Windows (CRLF) UTF-16 LE


output_weight.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
2759
false
true
2759
false
7724
false
5941
5743
false
false
2759
weight_union elapsedTime : 0.05899999999999999700
第 1 列，第 1 行 100% Windows (CRLF) UTF-16 LE


output_weight.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
2759
false
true
2759
false
7724
false
5941
5743
false
false
2759
weight_union elapsedTime : 0.05800000000000000300
第 1 列，第 1 行 100% Windows (CRLF) UTF-16 LE

平均所用的時間。

weight_union			height_union	
1	0.058		1	0.059
2	0.061		2	0.058
3	0.059		3	0.058
4	0.057		4	0.064
5	0.057		5	0.058
6	0.058		6	0.058
7	0.223		7	0.128
8	0.059		8	0.058
9	0.058		9	0.058
average	7.67E-02			6.66E-02

實驗如同預期 height_union 如果使用 collapsing_find 時的效能比較好。