Homework #3

資工二 陳盈如 B05902118

December 21, 2017

Reference

Problem 1

0.0.1 b05902074

0.0.2 b05902041

Problem 2

0.0.3 b05902074

0.0.4 b05902041

Problem 3

0.0.5 b05902041

Problem 5

0.0.6 助教周忠毅

Problem 5 - Compiler

(1)

```
1 typedef struct node {
2     id;
3     *left;
4     *right;
5     hash_value;
6     finish; //0:unfinished  1:finished
7 }node;  //The AST tree has already made.
8
9 typedef struct table {
10    id;
11    left_hash;
12    right_hash;
13 }table;
14
15 table hash_table[];
16
17 DFS(AST, *u) {
18    if (u->left == NIL)
19        left_value = 0;
20    else if (u->left->finish == 0) {
21        DFS(AST, u->left);
22        left_value = u->left->hash_value;
23    }
24    if (u->right == NIL)
25            right_value = 0;
26    else if (u->right->finish == 0) {
27        DFS(AST, u->right);
28        right_value = u->right->hash_value;
29    }
30    u->hash_value = hash(u->id, left_value, right_value);
31    hash_table[u->hash_value].id = u->id;
32    hash_table[u->hash_value].left_hash = left_value;
33    hash_table[u->hash_value].right_hash = right_value;
34    u->finish = 1;
```

```
35 }
36
37 convert(AST) {
38     DFS(AST, *root);
39 }
```

Listing 1: convert AST to DAG

(2)

Assume this algorithm cannot minimize the number of nodes, that is, there will be two or more nodes which can be merged together, but they take a space in hash_table respectively.
They take different places in hash_table, which means they have the different hash_values. However, if two nodes can be merged together, their hash_value should be the same.⇒ contradiction
Therefore, this algorithm can minimize the number of nodes.

(3)

Taking advantage of DFS function and the perfect hash table, we go through all nodes in the tree. At the mean time, we call DFS() exactily once for each edge in each node. Thus, DFS() is called E(the number of edges) times in total. The time complexity is O(E).
$\because$ E = N - 1;
$\therefore$ O(E) = O(N - 1) = O(N)
The time complexity is $\underline{\underline{\text{O(N)\#}}}$.

Problem 6 - Minimum Spanning Tree

(1)

By definition, any two vertexes in G is connected by a unique simple path, and there is no cycle in G. Then, G is a tree. Because roads connected all villages up with each other and the sum of edge weights should be minimized, if there is a circle, we can delete the largest edge to get a smaller weight, and it is still connected. In other words, T must be a tree, and its size is the number of villages in Wololo Kingdom.

(2)

$< pf >$
MST = a subtree of MST + a greedy choice of roads.
Let a subtree of MST is not optimal.
$\because$ subtree is not optimal
$\therefore$ 我們可以找到另一條未連接的 road 去取代已連接的 road 使 subtree 變成 optimal，新產生的 subtree + greedy choice 會比舊的 subtree + greedy choice 還要好，那舊的 subtree + greedy choice 就不是MST。$\Rightarrow$ $\boxed{\text{contradiction}}$

(3)

$< pf >$
Consider whether deleting e is a connected graph:
1.

If deleting e is a connected graph, e is the maximum weight edge that can be deleted to get the minimum cost in T.
2.

If deleting e is not a connected graph, e is the minimum cost edge to connect two trees T1 and T2. The **cut property** ensures that e is in T.

(4)

$E = \frac{|V|^2-|V|}{2}$

Kruskal's algorithm:

MAKE-SET = O(1) $\rightarrow$ *MAKE-SET* for $|V|$ items = $\underline{\text{O}(|V|)}$
Sort the edges into increasing order by weight = O($|V|^2 log|V|^2$) = $\underline{\text{O}(|V|^2 log|V|)}$
Take an edge in order, and **if** *FIND-SET(u)* != *FIND-SET(v)*, **do** *UNION(u, v)* = $\underline{\text{O}(|V|^2 log|V|)}$

The time complexity is $O(|V|) + O(|V|^2 log|V|) + O(|V|^2 log|V|) = \underline{\underline{O(|V|^2 log|V|)\#}}$

Prim's algorithm:

initialize every vertices takes $\underline{O(|V|)}$
MAKE-HEAP = $\underline{O(1)}$
INSERT = $O(1) \rightarrow$ *INSERT* for $|V|$ items = $\underline{O(|V|)}$
EXTRACT-MIN(Q) = $O(\log|V|)$
$\because$ **while** loop extracts min each time
$\therefore$ it will run $|V|$ times. $\rightarrow \underline{O(|V|log|V|)}$
DECREASE-KEY = $O(1)$
$\because$ each edge will go through once
$\therefore$ *DECREASE-KEY* runs E times in total $\rightarrow \underline{O(|V|^2)}$
The time complexity is $O(|V|) + O(1) + O(|V|) + O(|V|log|V|) + O(|V|^2) = \underline{\underline{O(|V|^2)\#}}$

(5)

$E = 3|V| - 6$ at most when $|V| \geq 3$

Kruskal's algorithm:

MAKE-SET = $O(1) \rightarrow$ *MAKE-SET* for $|V|$ items = $\underline{O(|V|)}$
Sort the edges into increasing order by weight = $\underline{O(|V|log|V|)}$
Take an edge in order, and **if** *FIND-SET(u)* != *FIND-SET(v)*, **do** *UNION(u, v)* = $\underline{O(|V|log|V|)}$
The time complexity is $O(|V|) + O(|V|log|V|) + O(|V|log|V|) = \underline{\underline{O(|V|log|V|)\#}}$

Prim's algorithm:

initialize every vertices takes $\underline{O(|V|)}$
MAKE-HEAP = $\underline{O(1)}$
INSERT = $O(1) \rightarrow$ *INSERT* for $|V|$ items = $\underline{O(|V|)}$
EXTRACT-MIN(Q) = $O(\log|V|)$
$\because$ **while** loop extracts min each time
$\therefore$ it will run $|V|$ times. $\rightarrow \underline{O(|V|log|V|)}$
DECREASE-KEY = $O(1)$
$\because$ each edge will go through once
$\therefore$ *DECREASE-KEY* runs E times in total $\rightarrow \underline{O(|V|)}$
The time complexity is $O(|V|) + O(1) + O(|V|) + O(|V|log|V|) + O(|V|) = \underline{\underline{O(|V|log|V|)\#}}$