# Algorithm Homework 10

Daoyu Wang

November 18

## Contents

**1 21.2-3** **ii**

**2 21.3-2** **ii**

**3 22.2-5** **ii**

**4 22.3-8** **iii**

**5 23.1-3** **iii**

**6 23.2-8** **iv**

# 1 21.2-3

According to the proof of Theorem 21.1 on **page 327** of the book, the time of the $n$ **UNION** operations is at most $O(nlog(n))$ which means that each of them takes amortized time $O(log(n))$.

Also, the time of the m **FIND-SET** and **MAKE-SET** operations is at most $O(m)$ which means that each of them takes amortized time $O(1)$.

# 2 21.3-2

**Algorithm 1** FIND-SET(x)

**Input:** $x$ indicates the given element
**Output:** $set(x)$ indicates which set $x$ belongs to

```
1: res ← ∅
2: Stack ← ∅
3: while x ≠ x.p do
4:     Stack.push(x)
5:     x ← x.p
6: end while
7: while Stack ≠ ∅ do
8:     y ← Stack.pop()
9:     y.p ← x
10: end while
```

# 3 22.2-5

According to the proof of Theorem 22.5 on **page 347** of the book, when the $BFS$ algorithm finishs, the $d$ value of each vertex $v$ is $\delta(s, v)$ which is the length of shortest path from $s$ to it. And the stortest path is determined when the graph is determined, not decided by the order of the vertices in the adjacency list. So $v.d$ has nothing to do with the order of the vertices in the adjacency list.

Use example 22.3 on **page 345** of the book to illustrate the **breadth-first tree** calculated by BFS can be different due to the order of the vertices in the adjacency list. Consider the vertices $\{w, t, x, u\}$, if $x$ is in front of $w$ in the adjacency list of $w$, then $x$ **ENQUEUE** before $w$, finally the $x$ connects with $u$ rather than $w$, so the **breadth-first tree** is different than the one in the book.

# 4 22.3-8

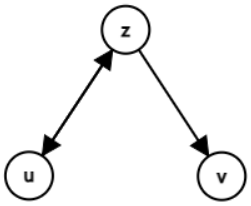


Figure 1: Counterexample 22.3-8

If we start from $z$, and $u$ is in front of $v$ in the adjacency list of $z$, **DFS** will find the path $z \to u \to z \to v$. Althouth there is a path $u \to z \to v$, in the **deep-first forest**, $u$ and $v$ are sibilings, $v$ is not a child of $u$.

# 5 23.1-3

**Lemma** We can proof that it exists a cut of the graph which contains just one edge of the minimum spanning tree.

Firstly, if there is a cut of the graph that contains no edge of the minimum spanning tree, then the cut splits the graph into two parts, and there is no connection between them. So the minimum spanning tree is not a tree, which is a contradiction.

Secondly, if there is a cut of the graph that contains at least two edges of the minimum spanning tree, there must be a cycle in the minimum spanning tree because two separate parts of the tree are actually two subtrees, and they are connected by at least two edges as mentioned above. Because of the existence of cycle, the minimum spanning tree is not a tree, which is a contradiction.

So it exists a cut of the graph which contains just one edge of the minimum spanning tree.

**Proof** According to this **lemma**, there is a cut which contains a specific edge $e$ of the minimum spanning tree. And the edge $e$ must be the lightest edge in the cut. Otherwise, if there is a lighter edge $e'$ in the cut, then the minimum spanning tree can be replaced by $e'$, which is a contradiction.

# 6 23.2-8

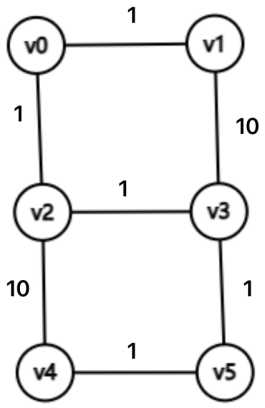


Figure 2: Counterexample 23.2-8

Obviously, the **MST** is $\{(v_0, v_1), (v_0, v_2), (v_2, v_3), (v_3, v_5), (v_5, v_4)\}$ and the total weight is 5.

However, if we split the graph into two parts $\{v_0, v_2, v_4\}$ and $\{v_1, v_3, v_5\}$, their respective **MST** is $\{(v_0, v_2), (v_2, v_4)\}$ and $\{(v_1, v_3), (v_3, v_5)\}$, the lightest edge in the cut is $(v_0, v_1)$. So if we use this divide-and-conquer algorithm, the final **MST'** is $\{(v_0, v_2), (v_2, v_4), (v_1, v_3), (v_3, v_5), (v_0, v_1)\}$, whose total weight is 23, larger than the **MST**.

So this divide-and-conquer algorithm is not correct.