Data Structure

Fall 2020, Homework #4 Solution

---

1 (a). 無法，因為在 merge tree 的時候只會 merge degree 相同的 tree，tree 的 node 數量只可能是 2 的次方。考慮到只有兩種 operations，insertion 只會產生 node 數量為 1 的 tree，而 find-andremove-min 執行 後會 merge degree 相同的 tree:

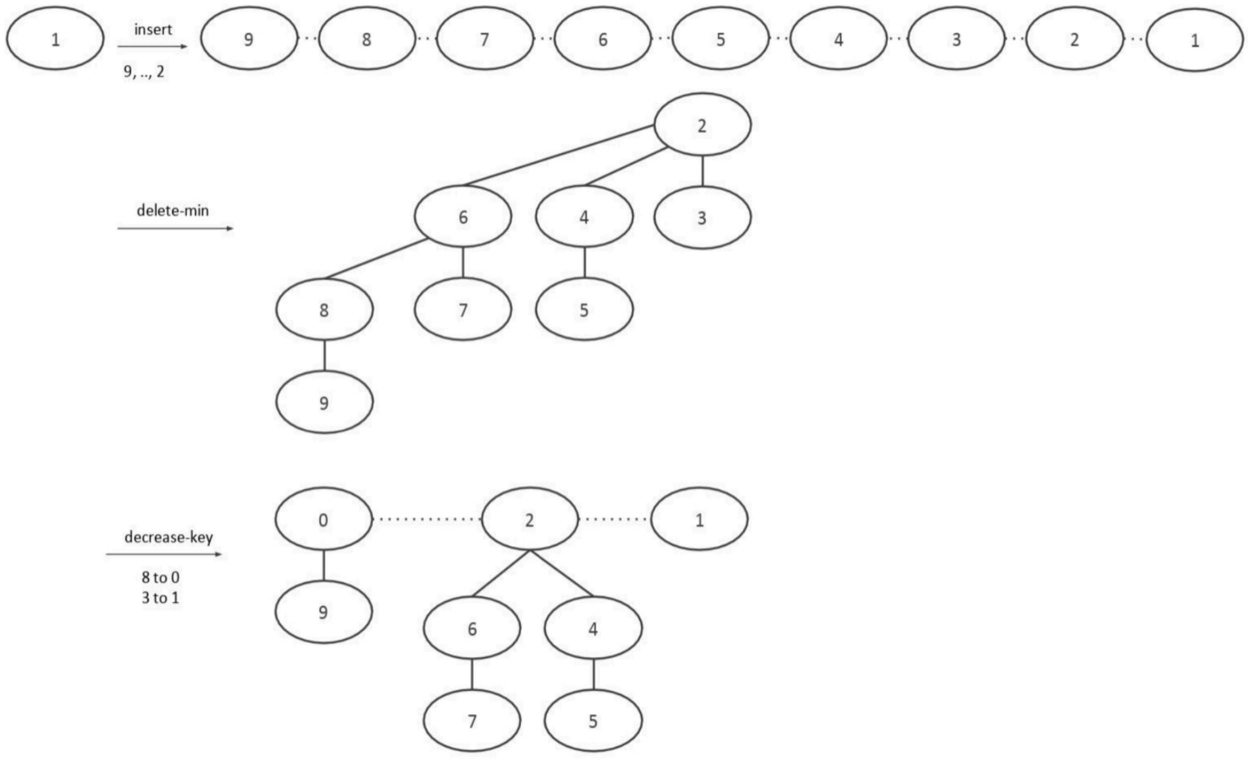
degree 為 1 的 tree 由 degree 為 0 的 tree merge 而來，node 數量為 2。

degree 為 2 的 tree 由 degree 為 1 的 tree merge 而來，node 數量為 4。

degree 為 3 的 tree 由 degree 為 2 的 tree merge 而來，node 數量為 8。

雖然 find-andremove-min 能減少 node 數量，但執行後 tree 會被拆成 degree 棵 tree，node 數量分別為 1, 2, 4, ... , 2^(degree-1) 個 node。node 數量依然會是 2 的次方。 因此只有這兩個 operation 無法產生 node 數量為 5 的 tree。

1 (b) 可以，按照以下步驟:



2 (a) The original union-find data structure utilizes weighted / rank union or union by height in order to achieve a better time per operation measurement. If we were to change the way union is done and make the minimum element be the root of the tree for its set, we might have to ignore the ranks / heights of the trees to be merged. This will then negatively impacts the time per operation.

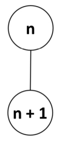
2 (b) Add an additional field to the tree nodes for storing the smallest member amongst its sub trees. In make-set, the smallest member is the node itself. In union operation, we need to compare the values stored in the two roots, and take the smaller one as the value for the new root. In find, one simply traverse the tree upwards to the root, and return the value stored in the root node. Note that although we have this field for every tree node, we only need to maintain the value for the root node.

3 (a)

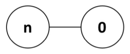
Algorithm:

```
        step 1. 把 n~1 依序放入 queue 中。
        step 2. insert(0)
        step 3. 從 queue 中拿 element e，insert(e)
        step 4. insert(n+1)
        step 5. remove-min()
        step 6. decrease-key(n+1, 0)
        step 7. 重複執行 step 3~6 直到 queue 為空
        step 8. remove-min()
```

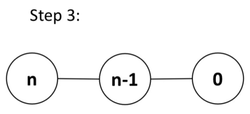
第一次執行 step 2, 3, 4, 5 後，會形成：

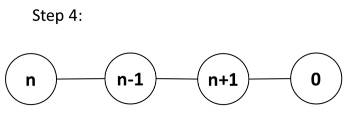


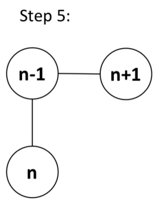
再執行 step 6 後會形成：

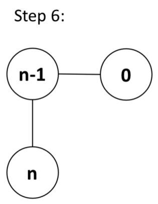


再重複一輪 step 3 ~ 6:









因此不斷重複 step 3~6 就可以產生一條長度為 n 的 chain，最後用 remove-min 把 0 去掉即可。

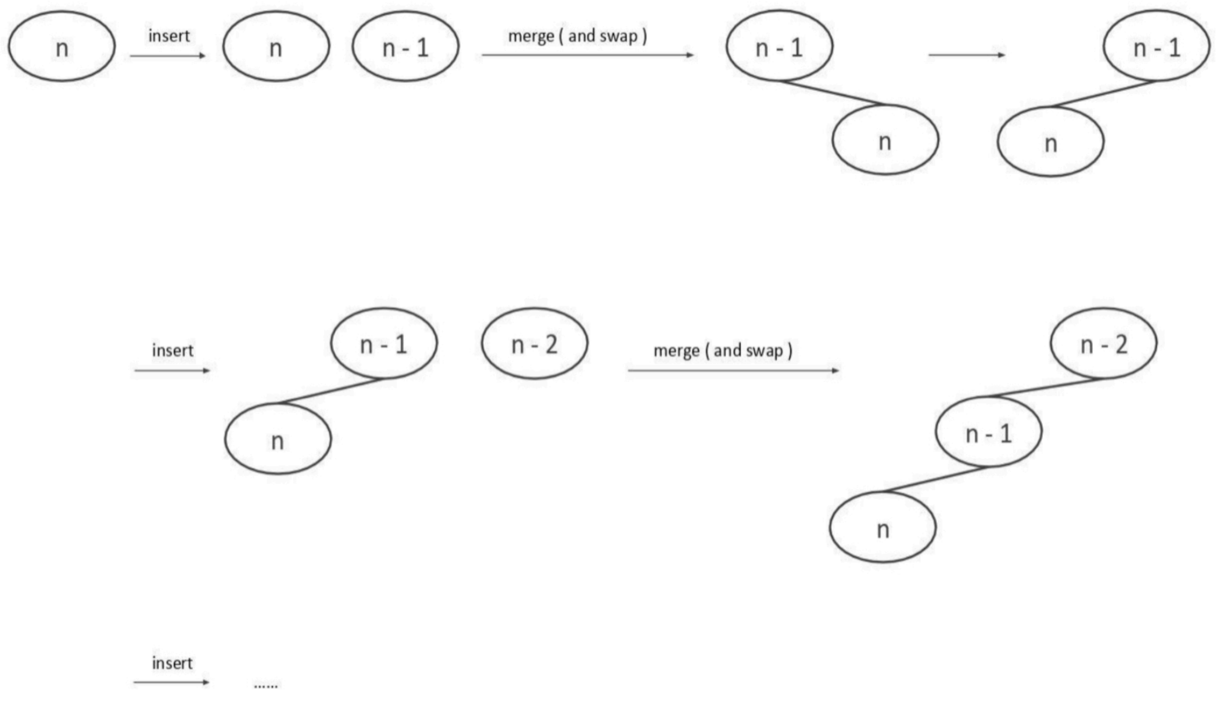
3 (b)

Algorithm:

step 1. 把 n~1 依序放入 queue 中，並拿出 element e 當作 t

step 2. 從 queue 中拿 並拿出 element e，與 t 合併

step 3. 重複執行 step 2，直到 queue 為空



4 (a) Similar to how we build mazes using union-find data structure. We start by putting each node into its own set. We then go through all the edges (by iterating through the adjacency lists of all nodes). For each edge $e = (v_1, v_2)$, perform union(v_1, v_2). When we're done, the number of sets in the union-find data structure is exactly the number of connected components in the graph.

4 (b) we performed n Make-Set operations, so it takes $n \times O(1) = O(n)$. Assume the union-find data structure we use implements weighted union and path compression. Since we performed m union operations, it takes $O\big(m \cdot \log^*(n)\big)$ (assuming that $m = \Omega(n)$). So the total time is $O\big(n + m \cdot \log^*(n)\big)$.

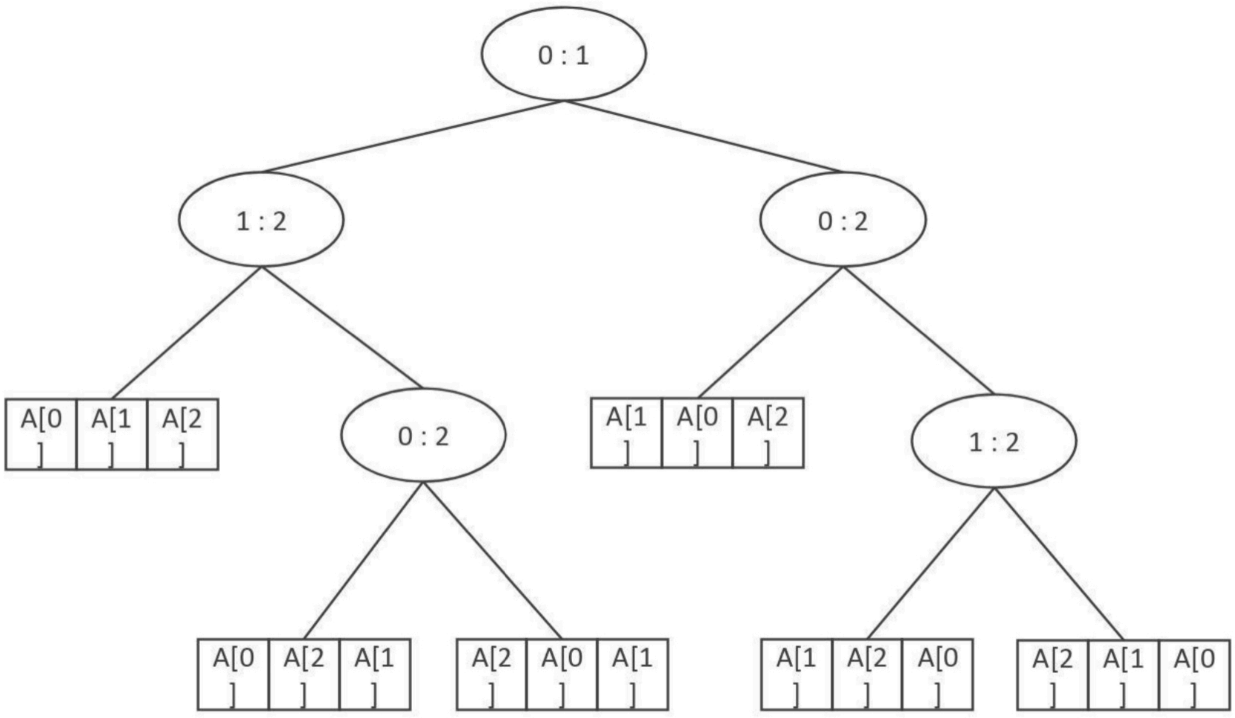
4 (c) $S = \varnothing$

```
For $v \in V$: {
  If $v$ is not marked {
      Mark $v$ and add $v$ to $S$.
      Perform depth-first-search starting from $v$ and mark all the nodes encountered on the way.
  }
Return: the number of elements in $S$.
```

Time complexity: $O(n + m)$

Since each node will be marked exactly once, marking the nodes takes $O(n)$. Once an edge is visited in the depth-first search, both nodes connected to the edge will be marked. Therefore the same edge will never be visited again by the algorithm. Hence the time it takes to complete the rest of the algorithm is $O(m)$.

5



0 : 1
1 : 2
0 : 2
A[0] A[1] A[2]
0 : 2
A[1] A[0] A[2]
1 : 2
A[0] A[2] A[1]
A[2] A[0] A[1]
A[1] A[2] A[0]
A[2] A[1] A[0]