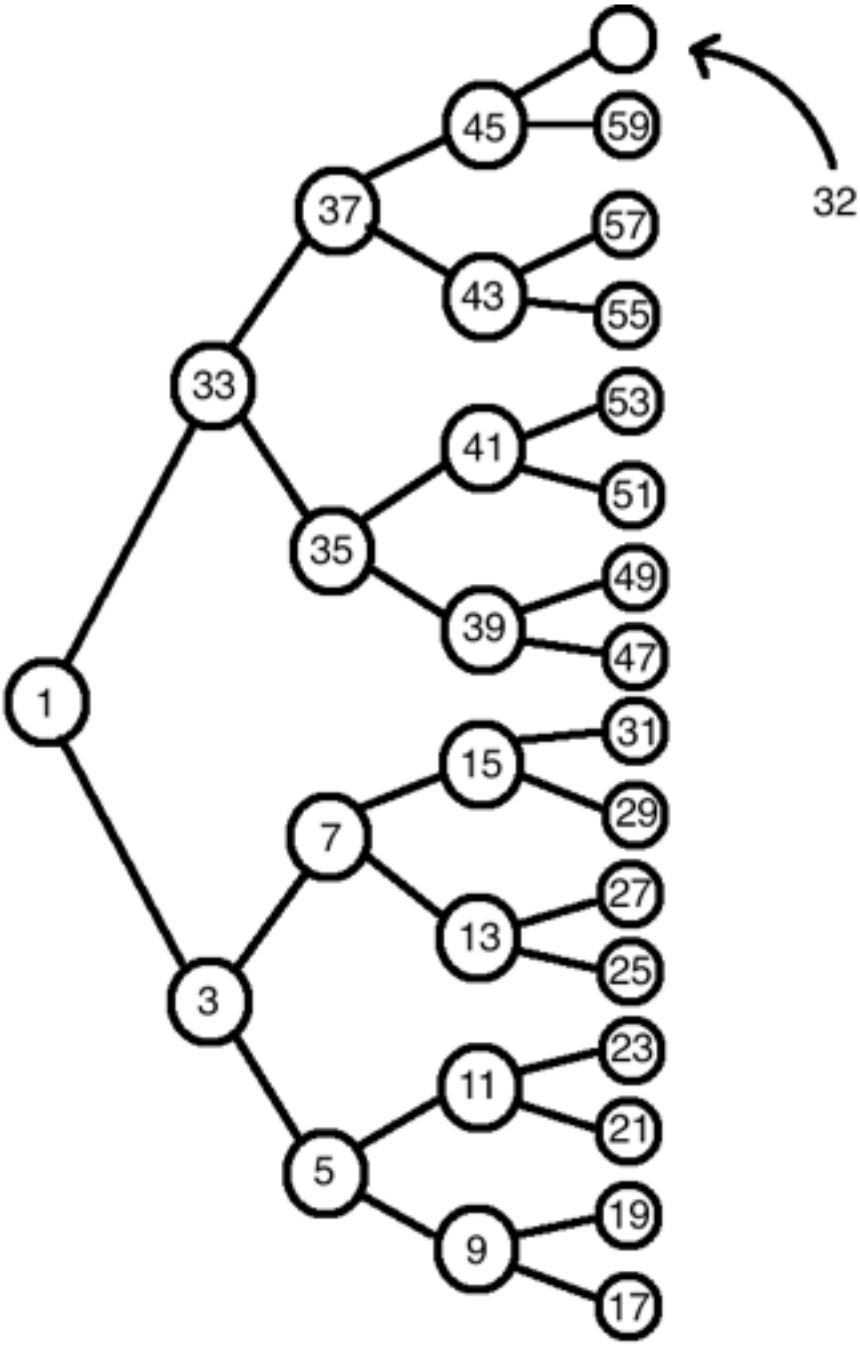
DSA HW5

B03902048 林義聖

1. Exercise R-8.24



2. Exercise C-8.4

```
STRUCTURE stack:
    data: max_priority_queue Q<Elem>
    data: top_priority

STRUCTURE Elem:
    data: priority
    data: value

PROCEDURE init(S):
    top_priority <- 0
    Q.clear()

PROCEDURE push(S,item):
    Q.push(Elem(top_priority,item))
    top_priority <- top_priority + 1

PROCEDURE pop(S):
    top_priority <- top_priority - 1
    return Q.pop().value
```

3. Exercise C-8.14

[pseudo-code]

```
FUNCTION findEntries(root, k)
    IF root.key() smaller than or equal to k THEN
        entry <- root.key()
    ELSE
        RETURN
    ENDIF

    IF root.hasRight() THEN
        findEntries(root.rightChild(), k)
    ENDIF
    IF root.hasLeft() THEN
        findEntries(root.leftChild(), k)
    ENDIF

    RETURN entry
END-FUNCTION
```

[/pseudo-code]

4. MinHash is used to quickly estimate the similarity of two sets. Generally, it's used to compare the similarity of two documents, while a document can be looked as a large set of words. Firstly, Jaccard similarity coefficient can be used to explain MinHash, while it is a commonly used indicator of the similarity between two sets. It looks like $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. When we use MinHash, we will use several hash function to compute and get many hash value on several subsets of two sets. If we use n methods to compute hash value and get k same value between two sets, k / n can be considered as the J(A,B) between two sets. This is MinHash. Following is the brief explanation of the implementation of MinHash on comparing two documents:
   - When it comes to MinHash, instead of comparing all of the words in two documents, we break down each document into a great deal of what are as known as shingles.

- Then, we use numerous hash algorithms to compute the hash value of each shingles, and store the smallest hash value computed by each hash algorithms.
- Next, compare all of the smallest hash value computed by various hash algorithms between two sets, and we will get the similarity between two sets.
- My reference: “Matt’s Blog - MinHash for dummies”, http://matthewcasperson.blogspot.tw/2013/11/minhash-for-dummies.html

5. [pseudo-code]

```
FUNCTION findDiffPos(str1,str2,length)
    IF length = 1 THEN
        RETURN str1
    ENDIF

    half <- length / 2
    IF postfixHash(str1,half) != postfixHash(str2,half) THEN
        IF length % 2 = 0 THEN
            RETURN findDiffPos(str1+half, str2+half, half)
        ELSE
            RETURN findDiffPos(str1+half+1, str2+half+1, half)
        ENDIF
    ELSE
        IF length % 2 = 0 THEN
            RETURN findDiffPos(str1, str2, half)
        ELSE
            RETURN findDiffPos(str1, str2, half+1)
        ENDIF
    ENFIF
END-FUNCTION
```

[/pseudo-code]

The time complexity of this algorithm is O(log n). Because the time complexity of postfixHash() is O(1), and as the pseudo-code above, binary search can be used to speed up the process and its time complexity is O(log n), the time complexity of findDiffPos() is O(log n).

6. My perfect hash function as following:

[pseudo-code]

```
FUNCTION hashFunc(String words)
    // ascii code
    f <- (int)words[0] - 97
    p <- (int)words[words.length()-1] - 97
    l <- words.length()
    RETURN (l * 89 * 89 + p * 91 * f) mod 81
END-FUNCTION
```

[/pseudo-code]

The result of the hash function above is range from 3 to 78, and all distinct. The output of the function is as following:
3, 6, 16, 17, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 42, 43, 45, 47, 51, 55, 56, 57, 58, 64, 65, 66, 67, 68, 72, 74, 76, 78

And we can use an integer array with size less than 80 to map each hash value into range from 0 to 31. The time complexity of this hash function is O(1). Its usage of memory is small enough, and it’s also computable.