Find Stepping Numbers in range [n,m]

DAA ASSIGNMENT-6 , GROUP 20

Harsh Mahajan
IIB2019001

Pradhuman Singh Baid
IIB2019002

Vasu Gupta
IIB2019003

***Abstract*—This Paper contains the algorithm to find all the stepping numbers in range [n, m]. A number is called stepping number, if all adjacent digits have an absolute difference of 1. 321 is a Stepping Number while 421 is not.**

I. Introduction

A number is called stepping number if all adjacent digits have an absolute difference of 1.
A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO).

II. Algorithmic Design

A. *Approach 1*

1. Its a brute force approach
2. Iterate over all number between [n,m] and checking number is stepping number or not.
3. Iterate through all digits of x and compare difference between value of previous and current digits, if difference >1 or $<$-1 then its not stepping number otherwise storing x in answer.
4. Print all number in answer which are stepping numbers between n and m.

B. *Approach 2*

1. This approach is based on graph traversing bfs.
2. We are using the fact that single digit number is stepping number.
3. Then we are generating graph as follow .
4. Every node in the graph represents a stepping number. there will be a directed edge from a node U to V if V can be transformed from U.
5. An adjacent number V can be: U*10 + lastDigit + 1 (Neighbor A) U*10 + lastDigit – 1 (Neighbor B)
6. then checking if node value is between n and m. which means this node is required stepping number and storing it into answer array.
7. Print all number in answer which are stepping numbers between n and m.

Algorithm 1: Finding Stepping Numbers

```
Input: n, m
Output: Prints Stepping numbers in [n,m]
Function SteppingNumbers(n,m):
    for i ← n to m do
        flag ← 1;
        prevd ← −1;
        j ← i;
        while j > 0 do
            cur ← j%10;
            if prev = -1 then
                prev ← cur;
            else
                if abs(prev-cur) != 1 then
                    flag ← 0;
                    break;
            prev ← cur;
            j ← j/10;
        if flag = 1 then
            print i;
```

III. Algorithm Analysis

A. *Approach 1*

For each number,to check that it is stepping number can be checked in some number of operation which is equivalent to constant time.since we are checking all numbers between n to m therefore execution time is $\propto m - n$. So the time complexity will be. O(m-n)

B. *Approach 2*

During bfs,once we encounter a number 'x' from the queue which is greater or equal to M, we don't need to push its stepping numbers in the queue as they will obviously again be greater than M. this gives you an upper bound. Now we are creating sort of 9 trees with root [1,9] respectively. for each node of the tree the the next child node has a number obtained after a multiplication factor of 10, mean taking jumps in powers of 10, which means a logarithmic sol with an upper bound M.so the number of nodes in the tree has to be log(M) moreover as its a tree number of edges are V-1, so the above approach would be log(M).

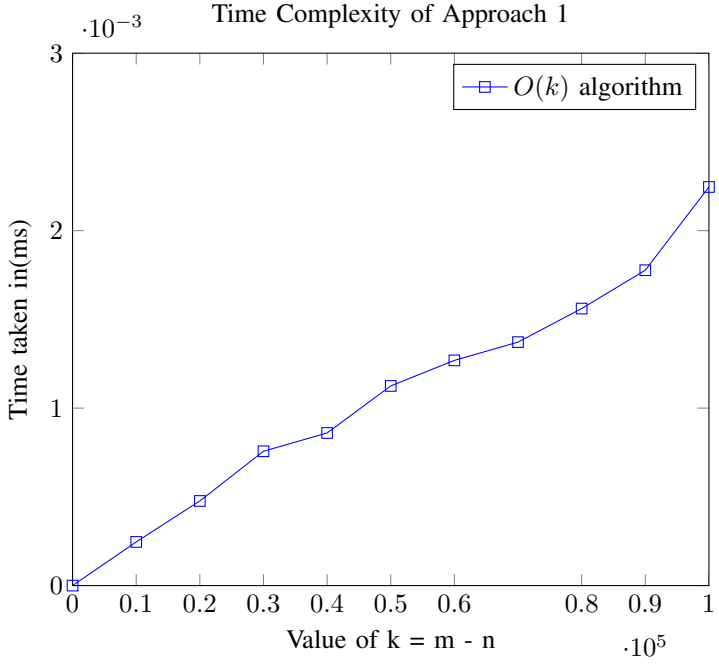
Algorithm 2: Finding Stepping Numbers

Input: n, m, num
Output: Prints Stepping numbers between n and m inclusive

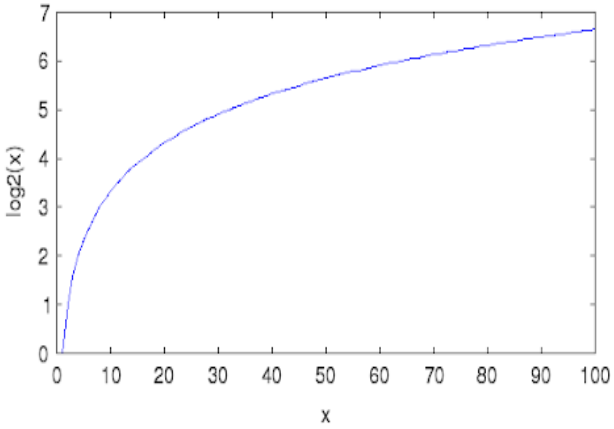
```
Function SteppingNumbers(n, m, num):
    queue <int> q;
    q.push(num);
    while !q.empty() do
        stepNum ← q.front();
        q.pop();
        if stepNum<=m and stepNum>=n then
            print stepNum;
        if num=0 or stepNum> m then
            continue;
        lastDigit ← stepNum%10;
        stepNumA ← stepNum×10+(lastDigit−1);
        stepNumB ←
         stepNum × 10 + (lastDigit + 1);
        if lastDigit= 0 then
            q.push(stepNumB);
        else if lastDigit=9 then
            q.push(stepNumA);
        else
            q.push(stepNumA);
            q.push(stepNumB);
```

IV. Experimental study

Time Complexity of Approach 1



Time Complexity for Approach 2



V. Space Complexity

We not storing anything so space complexity is almost constant

VI. Conclusion

Above two methods have different time complexities and meet to fulfill the problem statement. The order in which they are good can be listed as:
I. Approach 2
II. Approach 1
Based on the time complexities.

VII. References

1) 'Queue-Data-Structures',Wikipedia
2) 'Breadth First Search',GeeksforGeeks
3) Chirag Agarwal,'Stepping Numbers',GeeksforGeeks,2020