bloomfilter.py

Description:
This module will take 3 parameters as the input, that is, SNum (Data Item List), Bits (# of bits to represent the max number in the range query or the Data item list, whichever is max) and randK (the set of 7 private keys which is common for the whole tree)

Input: SNum, Bits, randK

Complexity: $O(n^2)$

Output: Returns the Bloom Filter of the respective node, also sends some important information with respect to the node like the vr, the unionSet which has the data of prefixes and the hashed values (this will later be used in the search algorithm)

Code:

```
import random # Used to generate random numbers
import hashlib # Used to generate Hash Values of the data

# This function will take 3 parameters the Data item list, number of bits and the private key list
# and generates the BloomFilter for that particular node in the tree.
def getBloomFilter(SNum, Bits, randK):
    storedSNUM = SNum[:] # Making a copy of the data itemss
    unionSet = [] # Used to store the union of the prefix set!
    S = [] # Main S list as per the algorithm
    bitLen = Bits
    preset = [] # Generates the prefix set for each data item
    appendFirst = [] # Temporary lists to store the intermediate values
    hashedAppendFirst = [] # Stores the final values of the 2 stage hashing process


    vr = random.randint(1, 1000) # Each node has a random value

    # This function is a accessor method to return the value of the private key for that node!
    def getVR():
        return vr

    # SHA-1 Hash function!
    def hashIt(s):
        s = str.encode(s) # Converts the String into bytes
        return hashlib.sha1(s).hexdigest()

    # Creates the prefixes
    def genPrefix(s, preSet):
        savedS = s # Copy of the S
        temp = []  # temp list to generate the Prefix Set of a binary value

        temp.append(s)
        s = list(s)

        for x in range(1, len(s) + 1):
            s[-x] = '*'
            temp.append(''.join(s))
        preSet += temp
```

```
    return preSet # Returns the set of prefixes for 's'

# Convert to fixed length binary!
string = '{0:0' + str(bitLen) + 'b}' # static BitLength

# This block converts the numerical elements in SNum to binary and adds them to 'S'
for n in SNum:
    bNum = string.format(n)
    S.append(bNum)

# Creating the prefix set!
for element in S:
    genPrefix(element, preset)

# Taking union of prefix set!
unionSet = [storedSNUM, list(set(preset))]
lol = []
mainLol = []

# For every element in unionSet create the hashed values of the elements for some N iterations!
for prefix in unionSet[1]:
    l = []
    hl = []
    lol = []
    for index, num in enumerate(randK):
        K = randK[index] + prefix # Appending the private key of the tree
        l.append(K)
        hK = hashIt(K) # First level of hash in the algorithm
        lol.append(hK)
        hK += str(vr) # Appending the private key of the node
        hhKvr = hashIt(hK) # Second level of hash in the algorithm
        hl.append(hhKvr)
    appendFirst.append(l)
    mainLol.append(lol)
    hashedAppendFirst.append(hl) # contains 2 Stage hashed values

M = (Bits - 1) * 10 * len(unionSet[1]) # Size required to create the bloom filter

bloomFilter = []

# Creating an array with default value as 0
m = 10000 * len(unionSet[1])

# Creating the bloom filter with 0 as the defualt value
for i in range(m):
    bloomFilter.append(0)

# Setting 1 to every index pointed by the mod operation!
for sett in hashedAppendFirst:
    for x in sett:
        a = int(x, 16)
        i = a % m
        bloomFilter[i] = 1 # Index pointed is set to 1

return bloomFilter, vr, unionSet, hashedAppendFirst # Returns data back to the function call
```