大数据原理与技术 Lab3

20337025 崔璨明

1.实验要求

Generate a bit stream of size (1,000,000):

1. Sampling a fixed proportion (10%) of the stream
2. Sampling a fixed-size sample (1,000)
3. Counting the number of 1s in the last N elements (N=50,000)

Analyze the result of each experiment.

2.实验过程

首先生成比特流:

```
LENGTH=1000000
# Generate the bit stream
stream = [random.randint(0, 1) for _ in range(LENGTH)]
```

为了模拟大数据流的情况，将一边遍历数据流steam，一边进行采样。

1、首先是按固定比例（10%）采样，遍历每个比特时，以概率10%进行选取:

```
# 1. Sampling a fixed proportion (10%) of the stream:
sampled_fix_proportion=[]
for bit in stream:
    p=random.random()
    if p<0.1:
        sampled_fix_proportion.append(bit)
```

2、接着是采样固定大小（1000），在遍历的过程中，以概率$p = \frac{size}{length}$选取每一个比特，若采样已满，则随机挑选一个进行替换:

```
# 2. Sampling a fixed-size sample (1,000)
sample_fix_size=[]
SIZE=1000
probability=float(SIZE)/float(LENGTH)
for bit in stream:
    p=random.random()
    # discard it
    if p>probability:
        continue
    if len(sample_fix_size)<SIZE:
        sample_fix_size.append(bit)
    else:
        choose=random.randint(0,SIZE-1)
        sample_fix_size.pop(choose)
        sample_fix_size.append(bit)
```

3、最后统计生成的比特流中最后 N（N=50000） 个元素中 1 的个数，采用DGIM算法进行统计，算法的具体实现如下，`DGIM`类维护一系列桶，利用桶对滑动窗口进行划分，每个桶保存以下信息：

- 桶的最右边位即最后进入流的位的时间戳；
- 桶的大小，即桶中1的个数，该数目位2的幂

数据的加入和数据结构的更新如下：

- 每一个新的位进入滑动窗口后，最左边一个位从窗口中移出（同时从桶中移出）；如果最左边的桶的时间戳是当前时间戳减去N（也就是说桶里已经没有处于窗口中的位），则放弃这个桶；
- 对于新加入的位，如果其为0，则无操作；否则建立一个包含新加入位的大小为1的桶；
- 由于新增一个大小为1的桶而出现3个桶大小为1，则合并最左边的两个桶为一个大小为2的桶；合并之后可能出现3个大小为2的桶，则合并最左边两个大小为2的桶得到一个大小为4的桶……依次类推直到到达最左边的桶

具体实现见`add()`函数和`update()`函数

对于统计最后 N（N=50000） 个元素中 1 的个数，即统计滑动窗口的1的个数，只需将滑动窗口中所有的bucket的size累加，但最早时间戳的bucket的size要乘以0.5，这是最早的那个bucket中有多少个1还留在滑动窗口中并不确定，假设其均匀分布。具体实现见`count()`函数。

```
class DGIM:
    def __init__(self, windowsize):
        self.windowsize=windowsize # window
        self.keysnum=int(math.log(self.windowsize,2))+1
        self.keylist=[]
        self.buckets = {}
        self.timestamp = 0  # init time stamp
        self.updatestep = windowsize  # updatestep <= windowsize
        self.updateidx = 0
        # init
        for i in range(self.keysnum):
            key=int(math.pow(2,i))
            self.buckets[key]=[]
            self.keylist.append(key)
    # update function
    def update(self):
        for key in self.keylist:
            if len(self.buckets[key])>2:
                self.buckets[key].pop(0)
                time_stamp=self.buckets[key].pop(0)
                if key!=self.keylist[-1]:
                    self.buckets[key*2].append(time_stamp)
            else:
                break
    # Counting the number of 1s in the last windows
    def count(self):
        nums=0
        first_stamp=0
        for key in self.keylist:
            if (len(self.buckets[key])>0):
                first_stamp=self.buckets[key][0]
        for key in self.keylist:
            for stamp in self.buckets[key]:
                if (stamp!=first_stamp):
                    nums+=key
```

```
                else:
                    nums+=0.5*key
        return nums
    # add a bit
    def add(self,var):
        self.timestamp=(self.timestamp+1) % self.windowsize
        for key in self.buckets.keys():
            for tstamp in self.buckets[key]:
                if self.timestamp == tstamp:
                    self.buckets[key].remove(tstamp)
        if var==1:
            self.buckets[1].append(self.timestamp)
            self.update()
        self.updateidx=(self.updateidx+1)%self.updatestep
        if self.updateidx==0:
            self.count()
```

3.实验结果与分析

生成的比特流见文件`stream.txt`，固定比例的采样结果见`sampled_fix_proportion.txt`，固定大小采样的结果见`sample_fix_size.txt`，统计最后 N（N=50000） 个元素中 1 的个数并输出，与真正的1的个数进行对比，计算误差，如下：

```
PS F:\VSCODE\arch_sysu> & E:/python/python.exe f:/VSCODE/arch_sysu/simple.py
Number of 1s in the original stream:  500038
Number of 1s in the sample(fixed proportion):  50047
Number of 1s in the sample(fix size):  502
Number of 1s in the last 50000 bits:  20806
Real number of 1s in the last 50000 bits::  24743
Error:15.911570949359415%
PS F:\VSCODE\arch_sysu>
```

实验结果表明，使用DGIM算法可以实现数据流的近似计数，并且相对误差很小（不大于50%），可以满足实际应用的需求，而且采集的样本中的1的个数的比例也和原始数据流中的一致。

DGIM算法在一定程度上解决了数据流采样和计数的问题，不仅可以在固定样本大小的情况下估计数据流中的1的个数，还可以在一定误差范围内计算数据流中最后N个元素中1的个数。同时，它的空间复杂度也比传统方法要小很多：该算法仅使用O(log2N)的空间，时间复杂度为O(logN)，而传统方法下空间复杂度O(N)，时间复杂度O(1)。

但是该算法虽然有着方法简单、空间复杂度低的优点，但也有一些限制，比如在计算最后N个元素中1的个数时，需要事先知道N的大小，并且在更新bucket时也需要事先知道窗口的大小。但是总的来说，DGIM算法是一种高效的算法，可以广泛应用于数据流计数、查询等场景。

4.总结与感想

通过这次实验，我复习了课上所学的知识，并能将其运用于实际问题中，这让我受益匪浅，对数据流采样和计数的问题有了更加深刻的理解。