大数据算法基础 实验2

张飞

2015210887

1. 问题描述

对于正整数流(数的范围为1到2^m),用DGIM算法估计流中大小为N的窗口内最近 k(1 ≤ k ≤ N)个整数的和。
其中, m = 8 , N = 100,000,000 , k = 50000000。

2. 实现思路

单个DGIM：首先实现单个DGIM算法，将其封装为一个类。对外提供输入[0,1]流的接口，以及估计最近k个中1的个数的接口。
m个DGIM求和：将一个输入的整数表示成m个2进制位，每位用一个DGIM流算法进行处理。当求k个最近值的和，最估计每个DGIM出现的1的各数，再乘以对应的权重。

3. 程序实现

3.1 单个DGIM算法实现

接口定义：

```
/*
 * DGIM.h
 *
 *  Created on: Dec 6, 2015
 *      Author: zhangfei
 */

#ifndef DGIM_H_
#define DGIM_H_

#include<iostream>
#include<vector>

typedef struct Bucket{
    unsigned int ts;    //该桶时间戳
    unsigned char j;    //该桶中的有2^j个1
} Bucket;

class DGIM {
private:
    std::vector<Bucket> buckets;    //所有的桶
    unsigned int ts;        //当前时间戳
    unsigned char r;            //每个桶最多有r个
    unsigned int N;             //时间窗口长度
public:
    DGIM(unsigned char r, unsigned int N) {
        this->r = r;
        this->N = N;
        ts = 0;
    }

    //向该流中追加一个0或1
    void append(unsigned char v);

    //估计最近k个值中1的个数
    unsigned int estimate(unsigned int k);

    //打印当前桶的状况
    void printBuckets();
};
#endif /* DGIM_H_ */
```

单个DGIM算法具体实现：

```cpp
/*
 * DGIM.cpp
 *
 *  Created on: Dec 6, 2015
 *      Author: zhangfei
 */

#include"DGIM.h"

using namespace std;

void DGIM::append(unsigned char v) {
    ts += 1;
    if (!buckets.empty()) {
        //判断最后一个的桶是否过期,如果过期将其覆盖删除
        if ((ts - buckets[0].ts) >= N) {
            for (int i = 1; i < buckets.size(); i++) {
                buckets[i - 1] = buckets[i];
            }
            buckets.pop_back();
        }
    }
    if (v == 1) {        //如果新加入元素为1，则对桶进行调整
        Bucket b = { ts, 1 };
        buckets.push_back(b);    //追加一个为1的桶
        vector<int> indexs;
        int count = 1;
        for (int i = buckets.size() - 2; i >= 0; i--) {
            if (buckets[i].j == buckets[i + 1].j) {
                count++;
                //如果相同桶的个数大于r，则进行合并
                if (count > r) {
                    buckets[i].ts = buckets[i + 1].ts;
                    buckets[i].j++;
                    indexs.push_back(i + 1);
                    count = 1;
                }
            }else{
                count=1;
            }
        }
        //删除已合并的桶
        for (int i = 0; i < indexs.size(); i++) {
            buckets.erase(buckets.begin() + indexs[i]);
        }
    }
}
```

```
unsigned int DGIM::estimate(unsigned int k) {
    unsigned int sum = 0;
    for(int i=buckets.size()-1;i >= 0;i--){
        //对时间戳与当前时间戳之差小于k的桶，算入估计值
        if((ts - buckets[i].ts) <= k){
            sum += (1<<(buckets[i].j-1));
        }else{
            break;
        }
    }
    return sum;
}

void DGIM::printBuckets() {
    for (int i = 0; i < buckets.size(); i++) {
        cout << "(" << buckets[i].ts << "," << (int) buckets[i].j
<< ")";
    }
    cout << endl;
}
```

3.2 m个DGIM算法求和

控制变量

```
#define lgMAX 8      //输入数2进制位长度
#define rMAX  5      //最多允许同时存在rMAX个桶
#define WINDOW_LENGTH 100000     //窗口长度
#define SLOT_LENGTH 1000         //当输入SLOT_LEGNTH个数时进行估计
#define ESTIMATE_LENGTH 50000    //估计最近的ESTIMATE_LENGTH个数的和
```

求和(读取文件)

```
#define BITV(x,i) ((x>>i)&1)

using namespace std;

int main(){
    ifstream fin("data.in",ios::in|ios::binary);

    //生成lgMAX个DGIM
    vector<DGIM> dgims;
    for(int i=0;i<lgMAX;i++){
        DGIM d(rMAX,WINDOW_LENGTH);
        dgims.push_back(d);
    }

    //读取随机数文件,先读入WINDOW_LENGTH个数，不估计
    unsigned int v;
    for(int i=0;i<WINDOW_LENGTH;i++){
        fin>>v;
        for(int j=0;j<lgMAX;j++){
            dgims[j].append(BITV(v,j));
        }
    }

    //每读入SLOT_LENGTH个数进行一次估计
    unsigned int count = 0;
    while(fin>>v){
        for(int i=0;i<lgMAX;i++){
            dgims[i].append(BITV(v,i));
        }
        count++;
        if(count == SLOT_LENGTH){
            unsigned int sum = 0;
            for(int i=0;i<lgMAX;i++){
                sum+= ((1<<i)*dgims[i].estimate(ESTIMATE_LENGTH));
            }
            cout<<sum<<endl;
            count = 0;
        }
    }

    fin.close();

    return 0;
}
```

求和(随机数流)

```
#define BITV(x,i) ((x>>i)&1)

using namespace std;

int main(){

    //生成lgMAX个DGIM
    vector<DGIM> dgims;
    for(int i=0;i<lgMAX;i++){
        DGIM d(rMAX,WINDOW_LENGTH);
        dgims.push_back(d);
    }

    //产生随机数，前WINDOW_LENGTH个数不估计
    srand(time(NULL));
    unsigned int v;
    for(int i=0;i<WINDOW_LENGTH;i++){
        v = (rand()%(1<<lgMAX));
        for(int i=0;i<lgMAX;i++){
            dgims[i].append(BITV(v,i));
        }
    }

    //每读入SLOT_LENGTH个数进行一次估计
    unsigned int count = 0;
    while(true){
        v = (rand()%(1<<lgMAX));
        for(int i=0;i<lgMAX;i++){
            dgims[i].append(BITV(v,i));
        }
        count++;
        if(count == SLOT_LENGTH){
            unsigned int sum = 0;
            for(int i=0;i<lgMAX;i++){
                sum+= ((1<<i)*dgims[i].estimate(ESTIMATE_LENGTH));
            }
            cout<<sum<<endl;
            count = 0;
        }
    }

    return 0;
}
```

4. 算法测试

4.1 准确度测试

测试思路：将所有测试数据写入到一个文件，例如shell脚本对文件某个范围内进行求和，获得标准结果。再将估计值于实际值进行比对，计算偏差百分比。
脚本(部分)：

```
#求和
head -$b data.in | tail -$k | awk '{sum+=$1} END
{printf("%d\n"),sum}'
#计算准确度
paste data.result data.out | awk 'function abs(x){return ((x>0.0)
? x : -x)} {a=(abs($2-$1)/$1);sum += a} END {print sum/NR}'
```

Test 1

测试条件：
总的随机数：1,000,000
窗口长度(WINDOW_LENGTH)：100,000
每多少个数进行估计(SLOT_LENGTH)：1,000
对最近多少个数进行估计求和：50,000
估计次数：900

测试结果：

允许桶重复数(rMax)	**2**	**3**	**4**	**5**	**10**	**25**	**50**
平均偏差值	0.329366	0.16407	0.0908717	0.0820722	0.0410942	0.0192747	0.0192747

从实验结果可以看出，当rMAX越来越大时，准确率越高。但到25时已经到准确率的极限。

Test 2
测试条件：
总的随机数：10,000,000
窗口长度(WINDOW_LENGTH)：1,000,000
每多少个数进行估计(SLOT_LENGTH)：10,000
对最近多少个数进行估计求和：500,000
估计次数：900

允许桶重复数(rMax)	**2**	**3**	**4**	**5**
平均偏差值	0.26156	0.130767	0.130767	0.065452

4.2 速度测试

测试思路：产生随机数流，不对随机数进行存储。在获得每次进行预测时，记录其预测时间。概平均实践包括产生SLOT_LEGNTH个随机数，并每次对桶进行调整的时间。预测100次，并取平均值。

Test 1

测试条件：
窗口长度(WINDOW_LENGTH)：1,000,000
每多少个数进行估计(SLOT_LENGTH)：10,000
对最近多少个数进行估计求和：500,000
估计次数：100

允许桶重复数(rMax)	**2**	**3**	**5**	**10**	**25**	**50**
平均估计时间(ms)	43.51	55.17	74.88	109.45	203.7	330.25

Test 2

测试条件：
窗口长度(WINDOW_LENGTH)：10,000,000
每多少个数进行估计(SLOT_LENGTH)：100,000
对最近多少个数进行估计求和：5,000,000
估计次数：50

允许桶重复数(rMax)	**2**	**3**	**5**	**10**
平均估计时间(ms)	461.2	591	805.05	1101.65

结论：有实验结果可以看出，当rMAX越大时对相同的预测，虽然准确度会提升，但是每次预测实践都会提升。因此实际应用时需要平衡两种情况。

5. 时间复杂度分析

假设：窗口长度为N，位数为m，桶最多重复r次，每n个数估计一次

- 一个DGIM对每一个位进行append的实践复杂度为：将所有桶便利一遍为$O(r*lgN)$，其中N为窗口长度。因为是利用vector数据结构，每一次删除操作需要花费的时间同样为$O(r*lgN)$，而最多删除lgN个元素。所以每次append操作的实践复杂度为$O(r*lgN^2)$
- 一个DGIM对最近的k个数进行估计时需要将所有桶扫描一遍，则复杂度为$O(r*lgN)$
- 共有m个DGIM算法进行估计，每一次估计过程中需要进行n次调整，和一次估计。所以每一次预测的实践复杂度为：

$$m*n*O(r*lgN^2)+m*O(r*lgN)$$