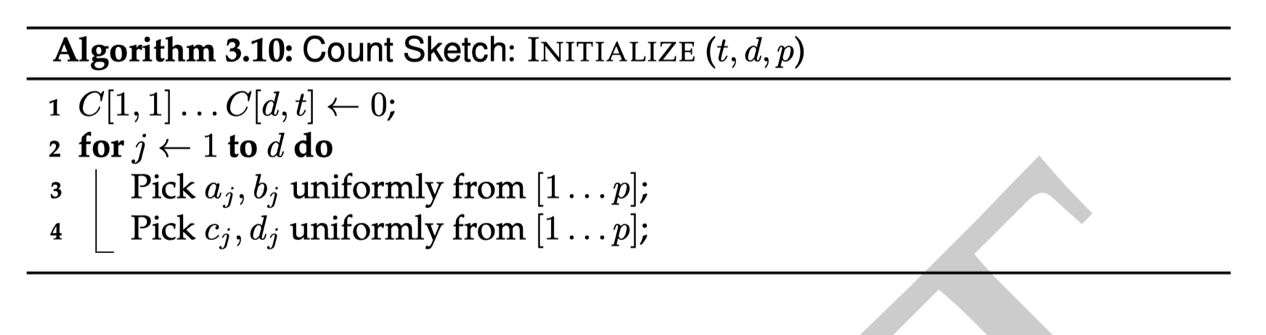
**第10周上机实验报告**

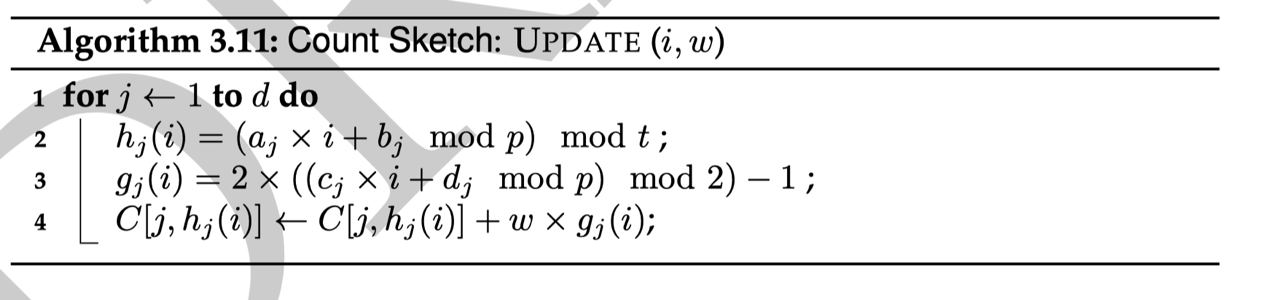
# 一、题目：

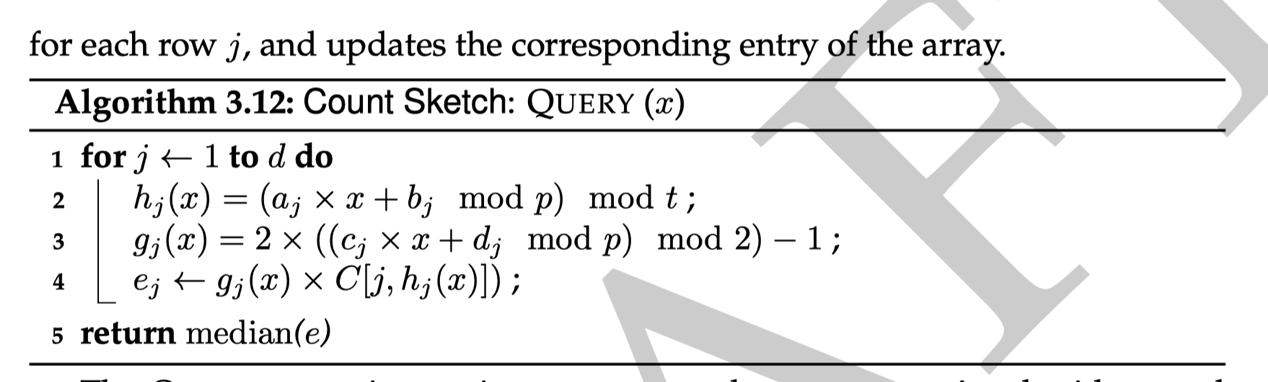
实现Count Sketch

# 二、算法思路：

具体描述算法的设计思想

****

****

****

# 三、程序设计框架：

## （1）核心函数的名称和功能

* init函数
  + 从{1,…,p}中均匀随机选取哈希函数中的参数a, b, c, d
  + 从输入文件中读取数据并存储下来

1. **void** init(**int** index)
2. {
3. //Pick a\_j,b\_j,c\_j,d\_j, uniformly from [1...p]
4. default\_random\_engine random(time(NULL));
5. uniform\_int\_distribution<**int**> dis1(1, p[index]);
6. **for** (**int** i = 0; i < d\_cs; i++)
7. {
8. a[index][i] = dis1(random);
9. b[index][i] = dis1(random);
10. c[index][i] = dis1(random);
11. d[index][i] = dis1(random);
12. //cout<<a[index][i]<<" "<<b[index][i]<<" "<<c[index][i]<<" "<<d[index][i]<<endl;
13. C.push\_back(vector<**int**>(t[index]));
14. }
16. //load data stream
17. fstream fin;
18. fin.open(file\_name[index]);
19. assert(fin.is\_open());
21. **int** a, b;
22. **while** (fin >> a >> b)
23. {
24. node.push\_back(a);
25. change.push\_back(b);
26. }
27. fin.close();
28. }

* update函数
  + 哈希函数h(x)将元素x映射到哈希表中
  + 哈希函数g(x)构造1或者-1
  + 用w\* 更新

1. **void** update(**int** index, **int** i, **int** frequency)
2. {
3. **for** (**int** j = 0; j < d\_cs; j++)
4. {
6. **int** h\_j\_i = (a[index][j] \* i + b[index][j]) % p[index] % t[index];
7. **int** g\_j\_i = 2 \* (((c[index][j] \* i + d[index][j]) % p[index]) % 2) - 1;
8. C[j][h\_j\_i] += g\_j\_i \* frequency;
9. //cout<<i<<" "<<C[j][h\_j\_i]<<endl;
10. }
11. }

* query函数
  + 哈希函数h(x)将元素x映射到哈希表中
  + 哈希函数g(x)构造1或者-1
  + 记录下所有哈希表中x对应的元素，返回其中的中位数

1. **int** query(**int** index, **int** i)
2. {
3. vector<**int**> e\_j(d\_cs);
4. **for** (**int** j = 0; j < d\_cs; j++)
5. {
6. **int** h\_j\_i = (a[index][j] \* i + b[index][j]) % p[index] % t[index];
7. **int** g\_j\_i = 2 \* (((c[index][j] \* i + d[index][j]) % p[index]) % 2) - 1;
8. //default\_random\_engine random(clock());
9. //uniform\_int\_distribution<int> dis1(0, 1);
10. //int g\_j\_i = dis1(random)\*2 - 1 ;
11. e\_j[j] = g\_j\_i \* C[j][h\_j\_i];
12. }
13. sort(e\_j.begin(), e\_j.end());
14. **return** e\_j[e\_j.size() / 2];
15. }

## （2）各核心函数之间的关系



## （3）输入输出的格式等

程序第十一行处的字符串数组中存储输入文件的路径，写成相对路径或绝对路径均可，默认数据集和代码在同一个文件夹下

1. string file\_name[3] = {"data\_100\_10.txt", "data\_10000\_1000.txt", "data\_1e+06\_10000.txt"};

程序第三十三行，main函数中index\_变量控制使用哪一个文件，编号对应字符串数组。

1. **const** **int** index\_ = 2;

output函数控制输出，默认输出到当前路径下，生成的输出文件为ansi.txt(i=0,1,2)

1. **void** output(**int** index);

# 四、实验结果说明：

## （1）实验结果截图

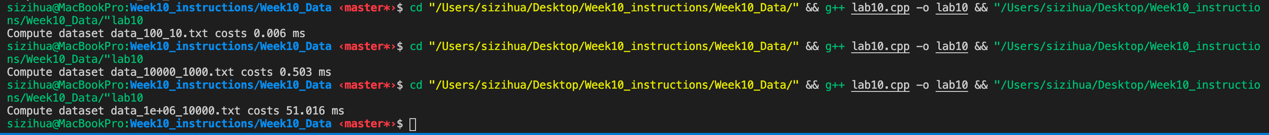


Figure :运行时间



Figure ：数据集0



Figure ：数据集1

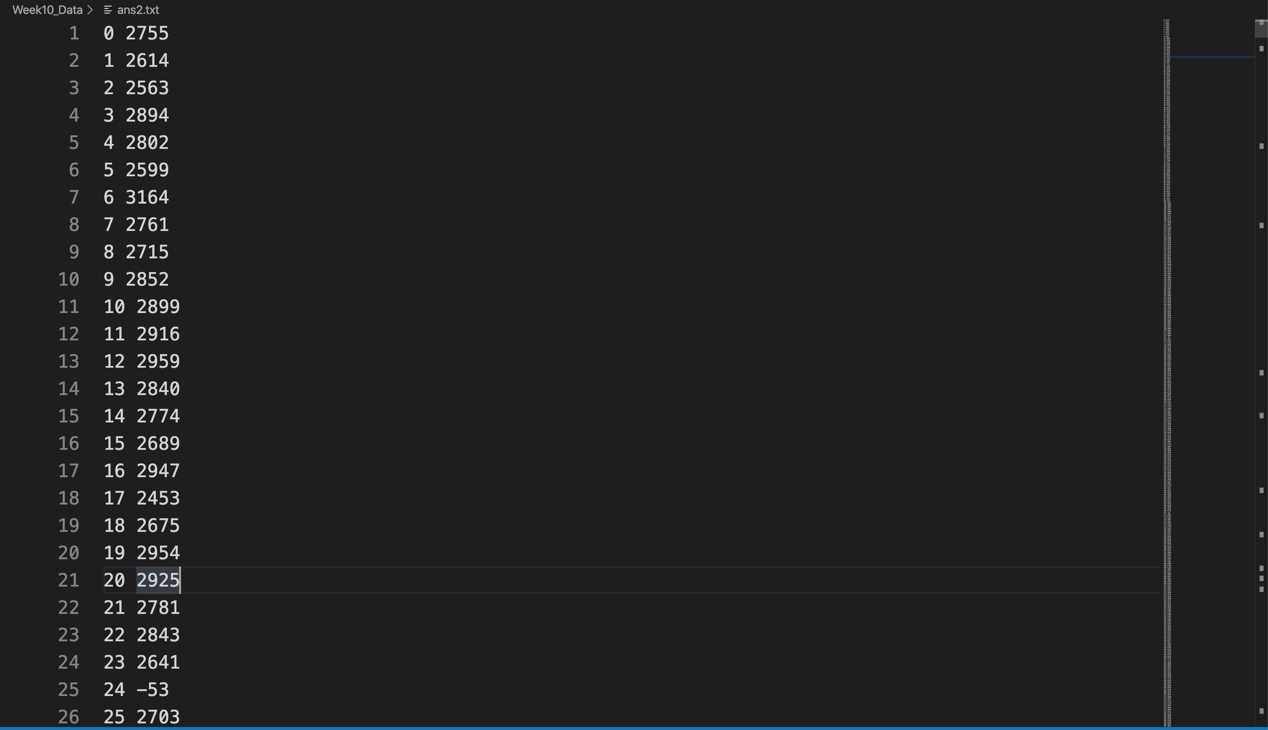


Figure ：数据集2

## （2）实验结果的分析

* 从运行时间上来看，三个数据集的运行时间确实是和数据流的长度线性相关
  + 第二个数据流的长度是第一个的100倍，运行时间也是其100倍
  + 第三个数据流的长度是第二个的100倍，运行时间也是其100倍
* 空间复杂度上来看，在d=3的情况下，表现不错
  + 但是上述运行结果中count sketch使用的总空间t\*d(t为一个哈希表的大小，d为哈希表的数量)是大于数据流中元素个数的，这样虽然取得了比较好的估计结果，但是还不如直接开一个元素个数N大小的数组去记录
  + 我在实验中尝试了很多次t\*d < N的组合，大部分情况下取得的结果并不是很理想，暂时没有相通为什么。检查了多次本程序，似乎也没有问题。猜测原因数据的方差比较大，导致误差较大。