埃德加·考特派代码思路-崔峻豪-电子科技大学

第一阶段比赛

第一阶段的比赛为小数据量并且只需要通过即可不需要考虑运行时间和内存的因素，所以仅需要把第二阶段的代码复制上去，改一下数据路径即可，所以主要说明第二阶段的代码。

第二阶段比赛

第二阶段的比赛由于要考虑到运行时间的影响，所以分为两个版本：
版本1：通用型，可以处理当数据量进一步增加到十几倍时的情况
版本2：仅针对当前数据量，让运行时间尽量快

下面先介绍版本1（通用型）：

版本1（通用型）：

代码如下：

```
#include<iostream>
#include <stdio.h>
#include<string.h>
#include <unordered_set>
#pragma warning(disable:4996)
using namespace std;
int main()
{
    //关闭cout输出流 加快cout的输出速度
    ios::sync_with_stdio(false);
    cout.tie(0);

    int SIZE = 300;//定义读文件时每一行的字节大小
    int cynum = 2;//通过设置这个数字 能够调节以多少取余 从而可以处理当数据量变大时的情况

    //用for循环来对文件按每一行的长度取余进行处理
    for (int cy = 0; cy < cynum; ++cy)
    {
        unordered_set<string>set(720000); //初始化一个set

        FILE* fin = fopen("/home/web/ztedatabase/2/input2.csv", "r");

        char buf[300];//定义一个char数组用于存放读取的每一行数据
        int id = 0; //用来记录读到了第几行
        while (fgets(&buf[0], SIZE, fin))
        {
            //当读到第10行之后 就从第3个字符开始 否则就从第2个字符开始 （这样能减少循环次数）
            int j = 2;
            if (id < 10)
            {
                j = 1;
            }
            while (buf[j] != ',')
            {
                ++j;
            }
            ++j;
            int lenj = strlen(buf) - j;
            if ((lenj & 1) == cy)
            {
                set.emplace(move(buf + j));
            }
            ++id;
        }
```

```
        FILE* fin1 = fopen("/home/web/ztedatabase/2/input1.csv", "r");

        char buf1[300];
        int idy = 0;
        while (fgets(&buf1[0], SIZE, fin1))
        {
            int j = 2;
            if (idy < 10)
            {
                j = 1;
            }
            while (buf1[j] != ',')
            {
                ++j;
            }
            ++j;
            int lenj = strlen(buf1) - j;
            if ((lenj & 1) == cy)
            {
                if (set.find(buf1 + j) == set.end())
                {
                    cout << buf1 + j;
                    set.emplace(move(buf1 + j));
                }
            }
            ++idy;
        }
    }
}
```

首先先讲解一下代码思路：
本通用型主要依靠读取的每一行数据的长度，按长度取余来实现分批次处理。先读取第2份数据（被比较的一份），按行读取，在读取时记录当前读取的行数，从而利用当前行数决定从第几个字符开始++判断数据第一个逗号的位置，以实现减少计算量的操作，每读取一份如果满足取余条件，就把读到的数据放到之前初始化好的set里，方便后续查重和去重。然后再读取第1份数据（用于比较的一份），对它的操作与之前的一样，当它满足取余条件之后，就在set里面查找，如果没找到，则输出，并放进set里面，防止重复，这样不断的操作下去，直到取余的for循环结束。

思路讲解：

```
ios::sync_with_stdio(false);
cout.tie(0);
//此代码用于加速cout的输出
```

此代码之所以能处理当数据量更大时的情况，是因为：读文件时按照一行一行的读取，然后计算每一行读取的数据长度（读取的数据是从第一个逗号开始），然后对长度取余，只要设置的取余的基数够大，就能够分为很多份来处理，从而能够处理数据量更大的情况。
因为本题的数据量仅需要对2取余就够了，所以设置的取余基数是2

```
int SIZE = 300;//定义读文件时每一行的字节大小
int cynum = 2;//通过设置这个数字 能够调节以多少取余 从而可以处理当数据量变大时的情况
```

然后就开始for循环，定义一个set用于去重，并且用于快速查找，因为set空间的成长会浪费时间，所以一开始直接把大小初始化好

```
unordered_set<string>set(720000); //初始化一个set
```

为了更进一步加快程序，分析数据格式，可以知道，数据开头由id和一个逗号组成，而这部分我们并不需要，所以可以考虑下面的算法：
当读取的是前10行时，从第2个字符开始取字符（也就是从逗号开始取字符，如2，这样就避开了前面的两个字符） 当读取的是后10行时，从第3个字符开始取字符（也就是从逗号开始取字符，如23，这样就避开了前面的三个字符）
这样能够在处理文件时，少很多的++运算
也就是以下代码的含义：

```
int id = 0; //用来记录读到了第几行
int j = 2;
if (id < 10)
```

```
{
    j = 1;
}
while (buf[j] != ',')
{
    ++j;
}
++j;
++id;
```

利用strlen函数计算出当前读出的数据长度，便于之后取余

```
int lenj = strlen(buf) - j;
```

一旦满足当前的取余条件，就把当前的数据放进set里面（这时是读取的第2份文件），等待读取第1份文件时用于排除相同的数据

```
set.emplace(move(buf + j));
```

然后读取第2份数据，与之前的步骤相同，一旦满足取余条件，就先在set里面查找，如果没找到，就输出出来，并放进set里面，防止重复

```
if (set.find(buf1 + j) == set.end())
{
    cout << buf1 + j;
    set.emplace(move(buf1 + j));
}
```

最后再进入下一次的for循环，根据取余基数进行下一次相同的操作，直到全部操作结束，这样就能实现目的。

下面先介绍版本1（针对型）：

版本1（针对型）：

代码如下：

```
#include<iostream>
#include <stdio.h>
#include<string.h>
#include <unordered_set>
#pragma warning(disable:4996)
using namespace std;


int idx[360000] = {};
int id1[360000] = {};
int SIZE = 300;

int main()
{
    ios::sync_with_stdio(false);
    cout.tie(0);

    unordered_set<string>set(720000);

    FILE* fin = fopen("/home/web/ztedatabase/2/input2.csv", "r");

    char buf[300];
    int id = 0;
    int idid = 0;
    while (fgets(&buf[0], SIZE, fin))
    {
        int j = 2;
        if (id < 11)
        {
            j = 1;
        }
```

```
    while (buf[j] != ',')
    {
        ++j;
    }
    ++j;
    int lenj = strlen(buf) - j;
    if ((lenj & 1) == 0)
    {
        set.emplace(move(buf + j));
        ++id;
    }
    else
    {
        idx[idid] = id;
        ++idid;
        ++id;
    }
}


FILE* fin1 = fopen("/home/web/ztedatabase/2/input1.csv", "r");
//FILE* fin1 = fopen("C:\\Users\\Administrator.DESKTOP-D4RD60B\\Desktop\\中兴\\input\\cjh11.csv", "r");

char buf1[300];
int idy = 0;
int ididy = 0;
while (fgets(&buf1[0], SIZE, fin1))
{
    int j = 2;
    if (idy < 11)
    {
        j = 1;
    }
    while (buf1[j] != ',')
    {
        ++j;
    }
    ++j;
    int lenj = strlen(buf1) - j;
    if ((lenj & 1) == 0)
    {
        if (set.find(buf1 + j) == set.end())
        {
            cout << buf1 + j;
            set.emplace(move(buf1 + j));
        }
        ++idy;
    }
    else
    {
        id1[ididy] = idy;
        ++idy;
        ++ididy;
    }
}
set.clear();

rewind(fin);

int idx2 = 0;
int ididx = 0;
while (fgets(&buf[0], SIZE, fin))
{
    if (idx[ididx] == idx2)
    {
        int j = 2;
        if (idx2 < 11)
```

```
            {
                j = 1;
            }
            while (buf[j] != ',')
            {
                ++j;
            }
            ++j;
            set.emplace(move(buf + j));
            ++ididx;
            ++idx2;
        }
        else
        {
            ++idx2;
        }
    }


    rewind(fin1);

    int idy2 = 0;
    int ididy2 = 0;
    while (fgets(&buf1[0], SIZE, fin1))
    {
        if (id1[ididy2] == idy2)
        {
            int j = 2;
            if (idy2 < 11)
            {
                j = 1;
            }
            while (buf1[j] != ',')
            {
                ++j;
            }
            ++j;
            if (set.find(buf1 + j) == set.end())
            {
                cout << buf1 + j;
                set.emplace(move(buf1 + j));
            }
            ++ididy2;
            ++idy2;
        }
        else
        {
            ++idy2;
        }
    }
}
```

首先先讲解一下代码思路：
由于本题数据量不是很大，取余基数为2就够了，然后考虑到在第一次循环时，已经计算了每一行的长度，并知道了哪一行处理过哪一行没有处理过了，所以在第二次循环时，就不需要再计算每一行的长度了，所以可以直接定义一个数组，用于存放第一次没有处理过的行号，在第二次循环时直接利用数组里记录的行号即可，这样能够节约时间。
然后其它操作类似于通用型的操作。
初始化两个数组，用于存放第一次循环时没处理的数据的行号：

```
int idx[360000] = {};
int id1[360000] = {};
```

当第一次循环结束时，要清理set里的元素，防止内存空间溢出，并把文件指针放到文件开头

```
set.clear();
```

```
rewind(fin);
rewind(fin1);
```

在第二次循环时，就不需要再计算当前数据长度并取余了，仅需要利用数组里存储的没处理的数据行号即可,这样可减少运行时间：

```
if (idx[ididx] == idx2)
...
++ididx;
if (id1[ididy2] == idy2)
...
++ididy2;
```

总结：本代码为了提高速度主要做的工作：

```
1、用此代码加速了cout的速度
    ios::sync_with_stdio(false);
    cout.tie(0);

2、初始化数组和set时直接指定空间大小，避免了set的数据插入时因空间成长而多出来的开销，并保证了set的空间大小足够多，使哈希碰撞的概率足够小
    int idx[360000] = {};
    int id1[360000] = {};
    unordered_set<string>set(720000);

3、分析了数据结构，按前10行和后10行的数据，来决定从第几个字符开始判断数据第一个逗号的位置
    int id = 0;
    int j = 2;
        if (id < 11)
        {
            j = 1;
        }
        while (buf[j] != ',')
        {
            ++j;
        }
        ++j;
    ```
    ++id

4、在传数据时，用的是传引用的方式（&），让程序运行更快
    while (fgets(&buf[0], SIZE, fin))

5、取余判断时利用与运算来加速判断为奇还是为偶
    if ((lenj & 1) == 0)

6、在往set容器里面放数据时，利用了emplace，并利用了右值表达式move，加快了放元素的速度
    set.emplace(move(buf + j));

7、设置了两个数组来存放第一次循环没处理的数据的行号，避免在第二次循环时的重复操作，提高了速度
    if (idx[ididx] == idx2)
    ...
    ++ididx;
    if (id1[ididy2] == idy2)
    ...
    ++ididy2;
```
```