链滴

PAT 甲级刷题实录——1009（写文章时又想到了改进方法）

作者：aopstudio

原文链接：https://ld246.com/article/1579231168128

来源网站：链滴

原题链接

https://pintia.cn/problem-sets/994805342720868352/problems/994805509540921344

思路

之前 1002 中问的是多项式的加法，这题则变成了多项式的乘法。前面的数据结构基本上一样，定义个多项式元素结构体 poly，以及存储多项式的两个 vector <poly> ，分别为 poly1 和 poly2。问题在结果的储存用什么数据结构。上次多项式的加法我用的是 vector，存储顺序类似于二路归并排序的序，但是这题不行了，因为乘积的指数并不能马上确定大小关系。

不能用 vector 的话有另外两种方法。

1. 使用链表，链表中的元素就是 poly 结构体，对此要在 poly 中加上指向下一个元素的指针。这种法的好处是大大节省空间，缺点是插入元素时需要一个个进行比较，时间复杂度提高了。而且 poly 的指针在 poly1 和 poly2 中没有用到，在存储输入数据的阶段反而浪费了空间。当然也可以将 poly1 和 poly2 都改成链表，但那样的话操作复杂度就又会上升。

2. 使用 float 类型的数组，数组中的元素代表系数，数组下标代表指数。这种方法的好处是插入数据不需要进行比较，时间复杂度降低了，缺点是会有大量 0 元素存在，浪费了存储空间。为了操作简便我们可以用 vector 来代替数组。

根据之前的实践经验，我们的策略是尽量用空间换时间，因此选择第二种方法。因此代码如下：

代码

```
#include <iostream>
#include <vector>
using namespace std;
struct poly
{
    int exp;    //指数
    double coe;    //系数
};
int main()
{
    int num1, num2;    //分别代表两个多项式所含元素的个数
    int max1, max2, min1, min2;    //记录两个多项式最大指数项对应的系数和最小指数项对应的系数
    vector<poly> poly1;    //第一个多项式
    vector<poly> poly2;    //第二个多项式
    vector<double> result;    //存储结果
    int resultSize;    //result的预留长度
    int realSize = 0;    //result中非零元素的个数
    int bias;    //result下标的偏移量
    int index;    //相乘后的结果在result中对应的下标
    cin >> num1;
    for (int i = 0; i < num1; i++)
    {
        poly p;
        cin >> p.exp >> p.coe;
        poly1.push_back(p);
        if (i == 0)
```

```
            max1 = p.exp;
        if (i == num1 - 1)
            min1 = p.exp;
    }
    cin >> num2;
    for (int i = 0; i < num2; i++)
    {
        poly p;
        cin >> p.exp >> p.coe;
        poly2.push_back(p);
        if (i == 0)
            max2 = p.exp;
        if (i == num2 - 1)
            min2 = p.exp;
    }
    bias = min1 + min2;
    resultSize = max1 + max2 - bias + 1;
    result.resize(resultSize);
    for (int i = 0; i < num1; i++)
    {
        for (int j = 0; j < num2; j++)
        {
            index = poly1[i].exp + poly2[j].exp - bias;
            result[index] += poly1[i].coe * poly2[j].coe;
        }
    }
    for (int i = 0; i < result.size(); i++)    //统计结果集中非零项个数
    {
        if (result[i] != 0.0)
            realSize++;
    }
    cout << realSize;
    for (int i = resultSize-1; i >= 0; i--)
    {
        if (result[i] != 0.0)
            printf(" %d %.1f", i + bias, result[i]);
    }
    return 0;
}
```

注意事项

这道题目有一个陷阱。如果你按照常规思路在计算完成后再统计 result 数组中非零项的数量，那么就会遇到这个陷阱。但是如果你想节省点时间在计算过程中根据当前添加的 result 位置是否为零（为零表这个指数对应的项第一次出现）来统计非零项的数量的话，就有可能出错，因为同个指数项的对应不同系数的乘积之和在多次加减后有可能为零，这样的话就不能将它统计进非零项的数量中了。具体码如下：

```
#include <iostream>
#include <vector>
using namespace std;
struct poly
{
    int exp;    //指数
```

```
    double coe;    //系数
};
int main()
{
    int num1, num2;    //分别代表两个多项式所含元素的个数
    int max1, max2, min1, min2;    //记录两个多项式最大指数项对应的系数和最小指数项对应的系数
    vector<poly> poly1;    //第一个多项式
    vector<poly> poly2;    //第二个多项式
    vector<double> result;    //存储结果
    int resultSize;    //result的预留长度
    int realSize = 0;    //result中非零元素的个数
    int bias;    //result下标的偏移量
    int index;    //相乘后的结果在result中对应的下标
    cin >> num1;
    for (int i = 0; i < num1; i++)
    {
        poly p;
        cin >> p.exp >> p.coe;
        poly1.push_back(p);
        if (i == 0)
            max1 = p.exp;
        if (i == num1 - 1)
            min1 = p.exp;
    }
    cin >> num2;
    for (int i = 0; i < num2; i++)
    {
        poly p;
        cin >> p.exp >> p.coe;
        poly2.push_back(p);
        if (i == 0)
            max2 = p.exp;
        if (i == num2 - 1)
            min2 = p.exp;
    }
    bias = min1 + min2;
    resultSize = max1 + max2 - bias + 1;
    result.resize(resultSize);
    for (int i = 0; i < num1; i++)
    {
        for (int j = 0; j < num2; j++)
        {
            index = poly1[i].exp + poly2[j].exp - bias;
            if(result[index]==0)
                realSize++;    //使用这个判断语句统计非零项数量
            result[index] += poly1[i].coe * poly2[j].coe;
        }
    }
    /*for (int i = 0; i < result.size(); i++)    //统计结果集中非零项个数
    {
        if (result[i] != 0.0)
            realSize++;
    }*/
    cout << realSize;
```

```
    for (int i = resultSize-1; i >= 0; i--)
    {
        if (result[i] != 0.0)
            printf(" %d %.1f", i + bias, result[i]);
    }
    return 0;
}
```

我一开始提交的几次都是部分正确，问题就是出在这，也就是第一次是非零项，但后来又加了几次变零了，结果又没有统计。不过这么一想我突然又有了新思路，如果我在执行加法语句后再加一行判断句，如果为 0 就令 realSize 减一会怎么样。现在这个想法完全是我正在写这篇文章的时候想到的。我来试一下……（尝试中）

结果证明是正确的，可以通过评测系统，这样的话就可以免去最后扫描整个 vector 统计非零项的过，又可以节省时间了。

现在经过改进后的代码如下：

```
#include <iostream>
#include <vector>
using namespace std;
struct poly
{
    int exp;    //指数
    double coe;    //系数
};
int main()
{
    int num1, num2;    //分别代表两个多项式所含元素的个数
    int max1, max2, min1, min2;    //记录两个多项式最大指数项对应的系数和最小指数项对应的系数
    vector<poly> poly1;    //第一个多项式
    vector<poly> poly2;    //第二个多项式
    vector<double> result;    //存储结果
    int resultSize;    //result的预留长度
    int realSize = 0;    //result中非零元素的个数
    int bias;    //result下标的偏移量
    int index;    //相乘后的结果在result中对应的下标
    cin >> num1;
    for (int i = 0; i < num1; i++)
    {
        poly p;
        cin >> p.exp >> p.coe;
        poly1.push_back(p);
        if (i == 0)
            max1 = p.exp;
        if (i == num1 - 1)
            min1 = p.exp;
    }
    cin >> num2;
    for (int i = 0; i < num2; i++)
    {
        poly p;
        cin >> p.exp >> p.coe;
        poly2.push_back(p);
```

```
        if (i == 0)
            max2 = p.exp;
        if (i == num2 - 1)
            min2 = p.exp;
    }
    bias = min1 + min2;
    resultSize = max1 + max2 - bias + 1;
    result.resize(resultSize);
    for (int i = 0; i < num1; i++)
    {
        for (int j = 0; j < num2; j++)
        {
            index = poly1[i].exp + poly2[j].exp - bias;
            if(result[index] == 0)
                realSize++;
            result[index] += poly1[i].coe * poly2[j].coe;
            if(result[index] == 0)
                realSize--;
        }
    }
    cout << realSize;
    for (int i = resultSize-1; i >= 0; i--)
    {
        if (result[i] != 0.0)
            printf(" %d %.1f", i + bias, result[i]);
    }
    return 0;
}
```

网上其他人的做法

中间卡在统计非零项过不去的时候我上网去搜索了一下其他人的做法。结果令我大失所望，竟然绝大数人都是在最开始创建了两个1000大小的数组存储多项式，一个2000大小的数组存储结果。还是那话，虽然尽量是用空间换时间，但也不带这么浪费空间的。我的两个多项式是用vector存储结构体，储结果的vector也是缩到尽可能小，size是两个多项式的最大指数和减去最小指数和。不过其中有一人的做法倒是挺有意思，虽然也是用的数组存储，不过他只用了一个数组存储第一个输入的多项式，输入第二个多项式的同时进行乘积运算，这样就省去了存储第二个多项式的数组空间，链接在此：https://blog.csdn.net/richenyunqi/article/details/78861275