北京大学
PEKING UNIVERSITY

期中习题课

10 宠物小精灵

题目描述

宠物小精灵的世界中，每位训练师都在收集自己的小精灵。但是，训练师的背包里只能同时携带6只小精灵。如果一位训练师抓到了更多的精灵，那么最早抓到的精灵将被自动传送到电脑中，保证持有的小精灵数量依然是6只。 训练师也可以随时从电脑中取出精灵，取出的精灵将从电脑中传送到训练师的背包里。取出的精灵同样被认为是最新的，导致背包中最早取出或抓到的精灵被替换到电脑中，训练师持有的精灵数量依然是6只。 初始状态下，所有训练师的背包中都没有小精灵，电脑中也没有任何训练师的精灵。

输入

输入数据包含多组测试。第一行一个整数T(1<=T<=20)，表示测试数据数目。
每组数据第一行有一个正整数N(1<=N<=20000)表示发生事件的数目。
接下来有N行分别表示发生的事件。
一共有两种事件：
C X, Y 表示训练师X抓到了精灵Y
T X, Y 表示训练师X试图从电脑中取出精灵Y。
X和Y都是长度在20以下的由字母或数字构成的字符串。

10 宠物小精灵

题目描述

小精灵的世界中同样存在着作恶多端的火箭队。他们试图利用电脑的漏洞，从电脑中取出本不属于自己的小精灵。因此，电脑需要识别并拒绝取出这种请求。注意，如果一只小精灵仅存在于训练师的背包中而未被传送至电脑，该训练师也不能取出这只精灵。相同训练师不会多次抓到相同名字的精灵。

输出

对于每次从电脑中取出小精灵的请求，输出一行。成功则输出Success，失败则输出Failed。

10 宠物小精灵

思路：

通过queue存储背包内的小精灵
通过set存储电脑内的小精灵

10 宠物小精灵

参考代码：

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    int T;
    cin >> T;
    for (int i = 1; i <= T; i++){
        map<string, queue<string> >bag;
        map<string, set<string> >computer;
        int num;
        cin >> num;
        for (int j = 1; j <= num; j++){
            char ord;
            string owner, pokeman;
            cin >> ord >> owner >> pokeman;
            if (ord == 'C'){
                queue<string>& tembag = bag[owner];
                set<string>& temcom = computer[owner];
                if (tembag.size() == 6){
                    temcom.insert(tembag.front());
                    tembag.pop();
                    tembag.push(pokeman);
                }
                else{
                    tembag.push(pokeman);
                }
            }
```

10 宠物小精灵

参考代码：

```
if (ord == 'C'){
    queue<string>& tembag = bag[owner];
    set<string>& temcom = computer[owner];
    if (tembag.size() == 6){
        temcom.insert(tembag.front());
        tembag.pop();
        tembag.push(pokeman);
    }
    else{
        tembag.push(pokeman);
    }
}
else if (ord == 'T'){
queue<string>& tembag = bag[owner];
set<string>& temcom = computer[owner];
if (temcom.count(pokeman) == 1){
    cout << "Success" << endl;
    temcom.erase(pokeman);
    tembag.push(pokeman);
    if (tembag.size() == 7){
        temcom.insert(tembag.front());
        tembag.pop();
    }
}
else if (temcom.count(pokeman) == 0){
    cout << "Failed" << endl;
}
            }
        }
    }
    return 0;
}
```

10 宠物小精灵

思考：

能否将set换成unordered_set来加快速度

扩展：

本质上是LRU缓存（Least Recently Used）的包装

可以进一步了解LFU缓存（Least Frequently Used）算法

11 改良过的CArray3d三维数组模板类

题目描述

```
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;
template <class T>
class CArray3D
{
// 在此处补充你的代码
};

CArray3D<int> a(3,4,5);
CArray3D<double> b(3,2,2);
```

11 改良过的CArray3d三维数组模板类

题目描述

```
void PrintA()
{
    for(int i = 0;i < 3; ++i) {
        cout << "layer " << i << ":" << endl;
        for(int j = 0; j < 4; ++j) {
            for(int k = 0; k < 5; ++k)
                cout << a[i][j][k] << "," ;
            cout << endl;
        }
    }
}
```

```
void PrintB()
{
    for(int i = 0;i < 3; ++i) {
        cout << "layer " << i << ":" << endl;
        for(int j = 0; j < 2; ++j) {
            for(int k = 0; k < 2; ++k)
                cout << b[i][j][k] << "," ;
            cout << endl;
        }
    }
}
```

11 改良过的CArray3d三维数组模板类

题目描述

```
int main()
{

    int No = 0;
    for( int i = 0; i < 3; ++ i )
        for( int j = 0; j < 4; ++j )
            for( int k = 0; k < 5; ++k )
                a[i][j][k] = No ++;
    PrintA();
    memset(a, -1, 60 * sizeof(int));        //注意这里
    memset(a[1][1], 0, 5 * sizeof(int));
    PrintA();
    for( int i = 0; i < 3; ++ i )
        for( int j = 0; j < 2; ++j )
            for( int k = 0; k < 2; ++k )
                b[i][j][k] = 10.0 / (i + j + k + 1);
    PrintB();
    int n = a[0][1][2];
    double f = b[0][1][1];
    cout << "****" << endl;
    cout << n << "," << f << endl;

    return 0;
}
```

11 改良过的CArray3d三维数组模板类

输入

无

输出

layer 0:
0,1,2,3,4,
5,6,7,8,9,
10,11,12,13,14,
15,16,17,18,19,
layer 1:
20,21,22,23,24,
25,26,27,28,29,
30,31,32,33,34,
35,36,37,38,39,
layer 2:
40,41,42,43,44,
45,46,47,48,49,
50,51,52,53,54,
55,56,57,58,59,
layer 0:
-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,
layer 1:
-1,-1,-1,-1,-1,
0,0,0,0,0,
-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,
layer 2:
-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,
layer 0:
10,5,
5,3.33333,
layer 1:
5,3.33333,
3.33333,2.5,
layer 2:
3.33333,2.5,
2.5,2,
****
-1,3.33333

11 改良过的CArray3d三维数组模板类

思路：

数组元素必须在内存中连续排列

CArray3d的operator[]的返回值类型

重载何种强制类型转换

参考代码：

```
class CArray3D
{
// 在此处补充你的代码
public:
    int x, y, z;
    T* p3;
    class CArray2D {
    public:
        T* p2;
        int col;
        CArray2D(T* p20, int y0) :p2(p20), col(y0) {};
        T* operator[](int r) {
            return p2 + r * col;
        }
    };
    CArray3D(int z0, int x0, int y0) :x(x0), y(y0), z(z0) {
        p3 = new T[x * y * z];//new在构造函数内部
    }
    CArray2D operator [](int layer) {
        return CArray2D(p3 + layer * x * y, y);
    }
    ~CArray3D() {
        delete[]p3;
    }
    operator void* ()
    {
        return (void*)p3;
    }
    // 在此处补充你的代码
};
```

11 改良过的CArray3d三维数组模板类

扩展：行优先与列优先

哪一个代码运行更快？

```
void PrintC()
{
    for(int i = 0;i < 3; ++i) {
        for(int j = 0; j < 4; ++j) {
            for(int k = 0; k < 5; ++k)
                cout << a[i][j][k] << "," ;
            cout << endl;
        }
    }
}
```

```
void PrintD()
{
    for(int k = 0; k < 5; ++k) {
        for(int j = 0; j < 4; ++j) {
            for(int i = 0; i < 3; ++i)
                cout << a[i][j][k] << "," ;
            cout << endl;
        }
    }
}
```

11 改良过的CArray3d三维数组模板类

扩展：行优先与列优先

在内存使用上，程序访问的内存地址之间连续性越好，程序的访问效率就越高

Fortune, Matlab等，采用列优先的数据存储方式

08 二进制数位复制并取反

题目描述

输入整数a b i j，把b从第i到j位(包括i,j)的二进制位全部取反，并填入a的i到j位中，a的其他位不变。输出a。 最右边一位是第0位。

输入

整数a b i j(范围不超过int)

输出

改变以后的a

08 二进制数位复制并取反

思路1：逐位操作

获取b的第k位，k∈[i,j]，根据该位修改a

08 二进制数位复制并取反

参考代码：

```
#include <iostream>
using namespace std;

int main(){
    int a, b, i, j;
    cin >> a >> b >> i >> j;
    for (int k = i, t; k <= j; ++k){
        t = ~b&(1<<k);
        a &= ~(1<<k);
        a |= t;
    }
    cout << a;
}
```

08 二进制数位复制并取反

思路2：整区间操作（基于mask）

将a的[i,j]区间全部设置为0
将b的[i,j]以外区间全部设置为0

08 二进制数位复制并取反

参考代码：

```
#include<iostream>
using namespace std;
int main(){
    int a=0,b=0,i=0,j=0;
    cin>>a>>b>>i>>j;
    int mask1=(1<<31>>(31-i));
    int mask2=~(1<<31>>(30-j));
    int mask=mask1&mask2;//第i到j位是1 其余位是0
    a=a&(~mask);//把a的第i到j位搞成0
    b=b^mask;//取反
    b=b&mask;//把b的除了i到j位搞成0
    a=a|b;
    cout<<a<<endl;
    return 0;
}
```

08 二进制数位复制并取反

位运算扩展：

- 交换两个变量的值：

```
a=a^b;
b=b^a;
a=a^b;
```

- 将最低位1置0：

```
x&(x-1)
```

- 仅保留最低位1：

```
x&(-x)
```

- 最接近的2的整数次幂(Java源代码）：

```
static final int tableSizeFor(int cap) {
    int n = cap - 1;
    n |= n >>> 1;
    n |= n >>> 2;
    n |= n >>> 4;
    n |= n >>> 8;
    n |= n >>> 16;
    return (n < 0) ? 1 : (n >= MAXIMUM_CAPACITY) ?
                                MAXIMUM_CAPACITY : n + 1;
}
```

07 求平均数的类真叫mean

题目描述

输入一系列正整数, 计算它们的平均值，结果去尾取整数。

输入

第一行是测试数据组数T
接下T行，每行一组数据。每行是若干个正整数，以0表示输入结束(0不算)。

输出

对每组数据，输出所有数的平均值，结果去尾取整数

07 求平均数的类真叫mean

题目描述

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
using namespace std;
class CMean {
// 在此处补充你的代码
};
```

```
int main(int argc, char* argv[]) {
    int  v;
    int t;
    cin >> t;
    while ( t -- ) {
        cin >> v;
        vector<int> vec;
        while (v) {
            vec.push_back(v);
            cin >> v;
        }
        int myAver = 0;
        for_each(vec.begin(), vec.end(), CMean(myAver));
        cout << myAver << endl;
    }
    return 0;
}
```

07 求平均数的类真叫mean

思路：

for_each函数接受何种参数

CMean内部应当设置哪些成员变量

CMean类需要实现何种成员函数

07 求平均数的类真叫mean

参考代码：

```
class CMean {
public:
    int* p;
    int count, sum;
    CMean(int& n) {
        p = &n;
        count = sum = 0;
    }
    void operator()(int x) {
        sum += x;
        count++;
        *p = sum / count;
    }
};
```

07 求平均数的类真叫mean

扩展

能否用lambda表达式来完成本题？

```
int myAver = 0;
int count=0, sum=0;
for_each(vec.begin(), vec.end(), [&](int x){
      sum += x;
      count++;
      myAver = sum/count;
});
cout << myAver << endl;
```

04 找第一个最小的

题目描述

写出 FindFirstLess 模板，用于寻找序列中小于某指定值的第一个元素

输入

第一行是测试数据组数T
接下来有2T行，每两行是一组测试数据
每组数据第一行开始是一个整数，表示这组数据有n项；接下来是一个字母，如果是'N'，表示这组数据都是整数，如果是'S'表示这组数据都是字符串
第二行就是n个整数，或者n个字符串

输出

对每组数据，输出第二行的前n-1项里面，第一个小于第n项的 。如果找不到，输出 "Not Found"

04 找第一个最小的

题目描述

```
vector<int> vi;
int n;
cin >> n; // size of the vector
int a,m;
for(int i = 0;i < n  - 1; ++i) {
      cin >> a;
      vi.push_back(a);
}
cin >> m;
vector<int>::iterator p = FindFirstLess(vi.begin(),vi.end(),m,less<int>());
if( p!= vi.end())
      cout << * p << endl;
else
      cout << "Not Found" << endl;
```

04 找第一个最小的

思路：

需要多少个类型参数
如何在函数内部操作迭代器（递增、大小比较）

04 找第一个最小的

参考代码：

```
template<class T1, class T2, class T3>
T2 FindFirstLess(T2 a, T2 b, T1 m, T3 op) {
    for (T2 i = a; i != b; ++i) {
        if (op(*i,m)) {
            return i;
        }
    }
    return b;
}
```

04 找第一个最小的

扩展：关于迭代器

```
template<class T1, class T2, class T3>
T2 FindFirstLess(T2 a, T2 b, T1 m, T3 op) {
    for (T2 i = a; i != b; ++i) {
        if (op(*i,m)) {
            return i;
        }
    }
    return b;
}
```

为什么不能写成i<b?

->Assertion failed: Vector iterators incompatible...

04 找第一个最小的

扩展：关于迭代器

Assertion failed: Vector iterators incompatible...

```
for (VectorType::iterator it = someVector.begin(); it != someVector.end();++it;)
{
    if (*it== value)
    {
        someVector.erase(it);
    }
}

for (VectorType::iterator it = someVector.begin(); it != someVector.end();++it;)
{
    if (*it== value)
    {
        it=someVector.erase(it);
    }
    else {++it;}
}
```

04 找第一个最小的

扩展：关于迭代器1

Assertion failed: Vector iterators incompatible...

```
for (VectorType::iterator it = someVector.begin(); it != someVector.end();++it;)
{
    if (*it== value)
    {
        someVector.erase(it);
    }
}

for (VectorType::iterator it = someVector.begin(); it != someVector.end();++it;)
{
    if (*it== value)
    {
        it=someVector.erase(it);
    }
    else {++it;}
}
```

04 找第一个最小的

扩展：关于迭代器2

Assertion failed: Vector iterators incompatible...

```
void merge_intervals(vector<int>& Interval1, vector< int >& Interval2, vector< int >& merged)
{
   vector< int >::iterator it1 = Interval1.begin();

   vector< int >::iterator it2 = Interval2.begin() ;

   while(it1 != Interval1.end() && it2 !=Interval1.end())
   { … }
}
```

09 积分图

题目描述

对于一幅灰度的图像，积分图中的任意一点(x,y)的值是指从图像的左上角到这个点的所构成的矩形区域内所有的点的灰度值之和。

输入

第一行两个整数，分别是图像的宽、高H, W
接下来H*W的矩阵，分别代表图像的每个像素值

输出

积分图中每个点的值, H*W的矩阵，每个像素之间用空格分开

04 找第一个最小的

题目描述

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
class IntegralImage{
// 在此处补充你的代码
};
int main(){
    int H, W;
    cin >> H >> W;
    int ** image = new int*[H];
    for(int i=0;i<H;++i){
        image[i]=new int[W];
    }
    for(int i=0;i<H;++i)
    for(int j=0;j<W;++j)
        cin >> image[i][j];
    IntegralImage it(H,W);
    for_each(image, image+H, it);
    for(int i=0;i<H;++i){
        for(int j=0;j<W;++j)
            cout<<it[i][j]<<" ";
        cout<<endl;
    }

}
```

04 找第一个最小的

常见错误

```
IntegralImage it(H,W);
for_each(image, image+H, it);
for(int i=0;i<H;++i){
    for(int j=0;j<W;++j)
        cout<<it[i][j]<<" ";
    cout<<endl;
}
```

一顿操作猛如虎->结果输出全0

std::for_each

Defined in header <algorithm>

```
template< class InputIt, class UnaryFunction >
UnaryFunction for_each( InputIt first, InputIt last, UnaryFunction f );
```

04 找第一个最小的

思路：

如何绕过for_each不能传引用的问题呢？

->用指针，使两个指针指向同一片内存区域
->构造函数中创建内存区域，复制构造函数中复制指针值

04 找第一个最小的

参考代码：

```
class IntegralImage{
public:
    int h,w,*a,i;
    IntegralImage(int _h,int _w){
        h=_h;w=_w;
        a=new int[h*w];
        i=0;
    }
    void operator()(int *b){
        for(int j=0;j<w;++j)
            a[i*w+j]=b[j]+(j?a[i*w+j-1]:0);
        if(i)
            for(int j=0;j<w;++j)
                a[i*w+j]+=a[(i-1)*w+j];
        ++i;
    }
    int * operator[](int x){
        return a+x*w;
    }
};
```