马上开始

35tang-C++竞赛系列四阶课程

三五堂

《35tang-C++竞赛系列四阶课程》

数据结构

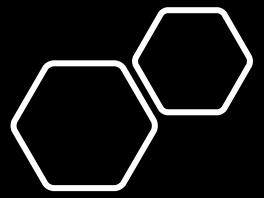
- C++尤其是STL提供了很多好用的数据结构
- 在程序中经常用到判断重复或者说是否出现过，这个时候我们经常使用flag计数方法或者set来做，也可以用map
- 有时候需要队列，可以用queue，或者priority_queue，deque也可以自己实现
- 有时候数组动态的，就用vector
- 如果这些类型的一个元素多个子元素组成，比如x，y坐标，就用struct或者pair（两个比较好）来实现。
- 也经常需要对他们进行排序，他们有自己的缺省排序顺序，对于简单的int string等类型可以方便的修改缺省排序顺序。对于自定义类型，比如struct，就需要自定义函数或者重载< 运算。

Set，优先队列等的自定义排序

重载<运算符

如果觉得太麻烦掌握不了，就避免放复杂类型

```
struct node{
   int x, y;
};
bool operator< (node a,node b)
{
   if(a.x == b.x)  return a.y >= b.y;
   else return a.x > b.x;
}
int main()
{
   priority_queue<node> pq;   //只传node，但是node结构体的<运算符已被改变
   for(int i = 1; i <= 5; i++)
      for(int j = 1; j <= 5; j++)
         {node tmp; tmp.x=i;tmp.y=j;pq.push(tmp);};
   while(!pq.empty())   {
      cout<<pq.top().x<<" "<<pq.top().y<<endl;
      pq.pop();
   }
   return 0;
}
```



USACO 2017 US Open Contest, Bronze Bovine Genomics

- http://www.usaco.org/index.php?page=viewproblem2&cpid=736

- 先看题，理解题目
- 用测试数据验证你是否真的理解了
- 注意数据量：n和m 100

分析题目

- Positions: 1 2 3 4 5 6 7 ... m
- Spotty Cow 1: A A T C C C A ... T
- Spotty Cow 2: G A T T G C A ... A
- Spotty Cow 3: G G T C G C A ... A

- Plain Cow 1: A C T C C C A ... G
- Plain Cow 2: A C T C G C A ... T
- Plain Cow 3: A C T T C C A ... T

该示例输入答案位1，因为只有位置2可以唯一识别。什么是唯一识别呢？

- 两个品种各给出n个奶牛，每一个奶牛有一个长度位m的字符串。要找位置1到m里面有多少个位置是满足条件的。也就是说这些位置的字符可以唯一辨认奶牛属于第一个品种还是第二个品种。
- position2是可以唯一识别两种奶牛的，因为如果字母是A或者G肯定是spotty cow，而如果是C，肯定是plain cow
- 其他位置不行，比如位置1，如果出现字母A，不知道是哪一个
- 结论：一个位置可以唯一识别的意思是这个字符只在有点或者没有点的奶牛中出现，在另外一种没有。

分析题目

- Positions: 1 2 3 4 5 6 7 ... m
- Spotty Cow 1: A A T C C C A ... T
- Spotty Cow 2: G A T T G C A ... A
- Spotty Cow 3: G G T C G C A ... A

- Plain Cow 1: A C T C C C A ... G
- Plain Cow 2: A C T C G C A ... T
- Plain Cow 3: A C T T C C A ... T

- 模拟
- 每一个位置挨个看，只需要记录两种奶牛各自的n个奶牛在这个位置出现过四个字母的哪几个，然后看是否有都出现的字母就知道答案了。
- 比如现在看1号位置。第一种奶牛出现了A，G，第二种出现了A，重复，所以这个位置不行。
- 思考：判断是否重复出现用什么方法？什么数据结构？

怎么做？

- N和M都不大（都小于100），而且只有A, C, G, and T四个字符出现。
- 这是一个判断重复的问题，数据量小可以用计数数组。
- 可以针对两种奶牛分别统计四个字母在每一个位置是否出现（用计数数组），这个也可以用二维数组表示，也可以用8个一维数组。

```
bool aflagspot[101]={0},cflagspot[101]={0},
        gflagspot[101]={0},tflagspot[101]={0};
bool aflagnospot[101]={0},cflagnospot[101]={0},
        gflagnospot[101]={0},tflagnospot[101]={0};
```

对于每一个位置，遍历所有奶牛，出现了某个字母就设置对应flag数组。结束遍历后，对于四个字母的四个flag数组分别遍历，如果两个奶牛的flag数组的那个位置都是true，说明重复了，这个位置不行。

```
//统计spot奶牛的四个字母在每一个位置是否出现过
for (int i=0;i<n;i++)
{
        fin>>inputstr;//读入字符串
        for (int j=0;j<m;j++)
        {
                t=inputstr[j];
                if (t=='A') aflagspot[j]=true;
                if (t=='C') cflagspot[j]=true;
                if (t=='G') gflagspot[j]=true;
                if (t=='T') tflagspot[j]=true;
        }
}

//统计nospot奶牛的四个字母在每一个位置是否出现过
for (int i=0;i<n;i++)
{
        fin>>inputstr;
        for (int j=0;j<m;j++)
        {
                t=inputstr[j];
                if (t=='A') aflagnospot[j]=true;
                if (t=='C') cflagnospot[j]=true;
                if (t=='G') gflagnospot[j]=true;
                if (t=='T') tflagnospot[j]=true;
        }
}

//枚举所有位置：看四个字母是否同时在两种奶牛都出现，其中只要有一个字母同时出现了这个位置就不行，就不能计数
int r=0;
for (int j=0;j<m;j++)
{
        if (aflagnospot[j]&&aflagspot[j]) continue;
        if (cflagnospot[j]&&cflagspot[j]) continue;
        if (gflagnospot[j]&&gflagspot[j]) continue;
        if (tflagnospot[j]&&tflagspot[j]) continue;
        r++;
}
fout<<r<<endl;
```

还可以怎么做?

- Positions: 1 2 3 4 5 6 7 ... m
- Spotty Cow 1: A A T C C C A ... T
- Spotty Cow 2: G A T T G C A ... A
- Spotty Cow 3: G G T C G C A ... A

- Plain Cow 1: A C T C C C A ... G
- Plain Cow 2: A C T C G C A ... T
- Plain Cow 3: A C T T C C A ... T

- 模拟
- 每一个位置挨个看。
- 对每一个位置：先把第一种奶牛出现的字母放到set或者设置对应的flag标记，搜索第二种奶牛的这个位置，每个字母和前面的set，或者flag去查找重复，如果出现过了，这个位置就不行。
- 当然map也是可以的，类似flag计数数组，但是慢一点点。

银牌

http://www.usaco.org/index.php?page=viewproblem2&cpid=739

N (1≤N≤500) and M (3≤M≤50)

Positions: 1 2 3 4 5 6 7 ... M

Spotty Cow 1: A A T C C C A ... T

Spotty Cow 2: G A T T G C A ... A

Spotty Cow 3: G G T C G C A ... A

Plain Cow 1: A C T C C C A ... G

Plain Cow 2: A G T T G C A ... T

Plain Cow 3: A G T T C C A ... T

- 还是4个字母，两种奶牛。
- N<=500,3=<M<=50
- 现在不找一个位置不同的，而是找三个位置组成的字符串有多少是两个奶牛是不同的。
- 1，2，3位置合起来就可以，因为两种奶牛组成的串没有重复的。5，6，7位置就不行，两种都出现了字符串CCA。这里字符串是有序的，比如ACT和CAT算不一样的。

怎么办？

- 三重循环去判断每一个3个位置的组合。

```
for (int i=1;i<=m;i++)
for (int j=i+1;j<=m;j++ )
for (int k=j+1;k<=m;k++ )
```

- 按照铜牌思路，把第一种n个奶牛出现的各种3个位置组合的字母保存起来（比如ACT,ATC,ATG...），看第二种n个奶牛出现的3个字母组合是否和上面第一种出现过的字母组合有重复的。如果这3个位置的组合不重复，那么计数++。
- flag计数数组？ 如果用计数方法来看是否重复，就需要对应一个数组下标，比如每一个状态需要是一个整数，而不能是类似ACT这样的字符串。

进制表示法：压缩唯一标记

- 因为只有三个位置，每一个位置是4个字母之一（四种状态），如果我们定义四个字母分别代表数字0，1，2，3。不超过4，如果三个位置分别代表一个四进制数字的每一位呢？
- 三位的四进制数：第一个位置的数字（0，1，2，3之一）*16+第二位的数字*4+第三位的数字。这个和就是一个唯一的数字，也就是说对于3个位置的任意字母组合，只会对应一个唯一的数字。而这个数字最大就是四进制的333，也就是十进制的3*16+3*4+3=63。这样，就可以继续使用flag计数的方法来做了。
- 遍历所有的3个位置的组合：初始化flag数组；对第一种每一个奶牛标记出现过的各种组合位置；第二种每一个奶牛生成组合位置去flag数组差看是否出现过（重复）,只要出现过这3个位置就不能用。
- 时间复杂度？O(m*m*m*n) 50*50*50*500大约$6*10^7$，极限了
- 如果用set或者map，更简单了，不需要进制压缩，可以直接把3个位置形成的字符串插入set或者作为map的key（类似flag计数数组的数组下标），很方便。但是得不了满分，用了set/map就要增加O(logn)的时间复杂度.。如果支持C++11，可以用unordered_map（用法和map类似就是不排序），这个就和flag计数的效率差不多了。

三重循环找所有的3个位置的组合，对于每一个组合：设置flag数组来标记重复，例如下面假设找到了1 3 5 位置，定义A位0，C为1，G为2，T为3。

注意：三个字母顺序不同，组合出来的数字也不同。

Positions: 1 2 3 4 5 6 7 ... M

Spotty Cow 1: A A T C C C A ... T

Spotty Cow 2: G A T T G C A ... A

Spotty Cow 3: G G T C G C A ... A

- 首先循环第一种奶牛n个奶牛字符串的1 3 5 位置的字符，修改flag如下：
- Spotty Cow 1: flag[0*16+3*4+1]=flag[29]=true
- Spotty Cow 2: flag[2*16+3*4+2]=flag[46]=true
- Spotty Cow 3: flag[2*16+3*4+2]=flag[46]=true

Plain Cow 1: A C T C C C A ... G

Plain Cow 2: A G T T G C A ... T

Plain Cow 3: A G T T C C A ... T

- 循环第二种奶牛所有的1 3 5位置，生成位置整数去flag数组找重复。Plain Cow 1: 0*16+3*4+1 =29 发现flag[29]为true，说明前面的奶牛出现过这样的组合，这个1 3 5 位置不行，退出循环。找下一个位置组合。

```
//i,j,k循环尝试所有的3个位置
for (int i=1;i<=m;i++)
for (int j=i+1;j<=m;j++ )
for (int k=j+1;k<=m;k++ )
{
      memset(flag,0,sizeof(flag));
      for (int l=1;l<=n;l++)
      {
      //对于每一个spottycow的i,j,k三个位置的字母一定会组合出来一个唯一的数字，把n个这样的奶牛的这个数字都记录在flag中
            t=getnum(spottycows,l,i)*16+getnum(spottycows,l,j)*4+getnum(spottycows,l,k);
            flag[t]=true;
      }
      bool ok=true;
      for (int l=1;l<=n;l++)
      {
      //对于每一个plaincow 的这3个位置，也一样得到一个唯一的数字，如果这个数字在前面n个spottycow的这3个位置出现过
      (flag做过标记)就说明这3个位置不行
            t=getnum(plaincows,l,i)*16+getnum(plaincows,l,j)*4+getnum(plaincows,l,k);
            if (flag[t]) {
                  ok=false;
                  break;
            }
      }

      if (ok) r++;
}
```

思考

- 程序中定义A为0，C为1，G为2，T为3。这样任意3个字母组合，最大也就是3*16+3*4+3=63。
- 如果出现的不是4个字母，而是26个呢？那就不能用四进制了，是26进制。每一个字母可以变成-'A'得到的数字，比如A对应0，Z对应25。这个组合的数字最大就是25*26*26+25*26+25= 17575。还好，也可以可以的。但是超过3个位置，再多就无法这样用了。这个时侯怎么判断重复？

金牌

```
Positions:     1 2 3 4 5 6 7 8

Spotty Cow 1: A A T C C C A T
Spotty Cow 2: A C T T G C A A
Spotty Cow 3: G G T C G C A A

Plain Cow 1:  A C T C C C A G
Plain Cow 2:  A C T C G C A T
Plain Cow 3:  A C T T C C A T
```

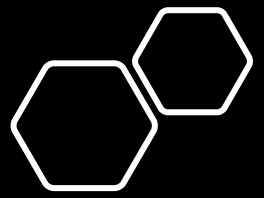
- 现在N<=500,3=<M<=500　M变大了。而且结果要求找到最短的可以识别的位置长度。注意，是连续的。上面位置238合起来的位置也不重复，但是不连续，不算。位置2345算。

http://www.usaco.org/index.php?page=viewproblem2&cpid=741

还可以flag么？

- 题目启发了我们，是连续字符串。那么就可以按照字符串长度去遍历，长度从小到大，**对于每一个长度的每一个开始位置**，n个奶牛就有最多n个子串，**我们要做的就是比较两种奶牛同一个长度同一个位置开始的n个子串之中有没有重复的。这个和银牌一个思路。**
- 但是这次要比较的位置不是固定长度的，最多500，如果用进制压缩数字，难道要500位？所以flag计数标记的方式不行。
- 不能flag计数，用什么？用set或者map。 对于每一个长度对应的每一个开始位置，把第一种奶牛形成的n个字符串插入set，然后第二种每一个奶牛的字符串去这个set查找一下是否重复就好了。
- 注意：set/map的插入查找时间复杂度都是O(logN)。总的时间复杂度m(字符串长度)*m（开始位置）*n（n个奶牛）*logn（插入set或者查找set）=O(m*m*n*logn)。 如果这里不用set用数组，时间复杂度就是O(n*n*m*n)了。

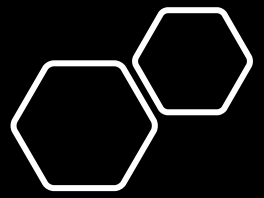
```
set <string> flagset;
int i;
bool findpos;
for (i=1;i<=m;i++)//枚举子串长度
{
      for (int j=0;j<=m-i;j++ )//枚举子串开始位置
      {
            findpos=true;
            flagset.clear();

            for (int l=1;l<=n;l++)//第一种的n个奶牛的子串都入set
            {
                  flagset.insert(spottycows[l].substr(j,i));
            }
            for (int l=1;l<=n;l++)// 第二种的n个奶牛的每一个子串去检查第一种出现过没有
            {
                  if  (flagset.count(plaincows[l].substr(j,i))!=0) {
                        findpos=false;
                        break;
                  }
            }
            if (findpos) break;
      }
      if (findpos) break;

}

cout<<i<<endl;
```



还是不行

最值问题？

- O(m*m*n*logn)大约是10的9次方了。
- 怎么办？
- 注意：我们要找的是符合条件的最小字符串长度，那么这个长度有没有单调性？能不能二分？能！ 因为肯定是越大越容易满足，如果一个长度k的可以满足条件，那么大于k的长度就不用去试了肯定可以（如果k长度子串都不相同，那么加上其他字符就算都一样，新的子串还是不相同的），去找小于k的有没有更合适的。
- 新的时间复杂度O(logm*m*n*logn)大约10的7次方



简单浏览程序

- 标准的二分法
- 课后看例题和练习里面的源程序
- 例子程序并没有用前面介绍的模板，比如对于每一个答案，没有用一个专门的函数checkok去判断这个答案是否可以，而是把判断逻辑和二分法放在了一起，你可以对比一下上节课的例题和作业，是不是用函数包装，用模板更容易理解？如果可以，自己试着重新做一下？

```
bool checkok(int mid)//检查长度mid是否符合要求
{
        for (int j=0;j<=m-mid;j++ )//枚举开始位置
        {
                set <string> flagset;
                bool isok=true;
                for (int l=1;l<=n;l++)//第一种的n个奶牛的子串都入set
                        flagset.insert(spottycows[l].substr(j,mid));
                for (int l=1;l<=n;l++)// 第二种的n个奶牛的每一个子串去检查第一种出现过没有
                        if  (flagset.count(plaincows[l].substr(j,mid))!=0)
                                {isok=false;break;}
                if (isok) return true; //当前j位置是可以的
        }
        return false;
}



//主程序中的标准二分模板
int leftbound=1;
int rightbound=m;
int mid,ans=m;
while (leftbound<=rightbound)
{
        mid=(leftbound+rightbound)/2;
        if (checkok(mid)) {
                rightbound=mid-1;
                ans=mid;
        }
        else leftbound=mid+1;

}
fout<<ans<<endl;
```

常用数据类型—map类型

map类型存放的是键值对，就是说根据key的值找value.

比如:

字符串”abc”配对数字100

字符串”ac”配对数字120

字符串”ad”配对数字99

..

给出了很多个这样的键值对，也就是说字符串是key，后面一个整数是value。现在如果给出一个字符串，怎么才能最快的查找到这个字符串对应的整数呢？用map。

和set类似，内部也是用二叉树(二叉查找树,红黑树)实现的，所以查找效率也是O(logn).比flag计数数组慢一点，但是如果C++11支持，里面的unordered_map就和flag计数数组效率几乎一样了。

map也是排序的，缺省从小到大

看程序学习用法

[]的用法特别类似数组。

```
#include <iostream>
#include <map>
using namespace std;
 int main ()
{
    map<char,int> mymap;
    map<char,int>::iterator it;
    mymap['a']=50;
    mymap['b']=100;
    mymap['c']=150;
    mymap['d']=200;

    it = mymap.find('b');
    if (it != mymap.end())     mymap.erase (it);
    // print content:
    cout << "elements in mymap:" << '\n';
    cout << "a => " << mymap.find('a')->second << '\n';
    cout << "c => " << mymap.find('c')->second << '\n';
    cout << "d => " << mymap.find('d')->second << '\n';
    return 0;
}
```

遍历 和begin(),end() []可以用来赋值，可以用来访问。

```
// map:用法非常类似flag计数数组，遍历不太一样
#include <iostream>
#include <map>
 int main ()
{
    map<char,int> mymap;
    mymap[‘b’] ++;
    mymap['a’] ++;
    mymap['c’] ++;
    cout << mymap[‘c’]; //找到就是出现的次数
    cout << mymap['d']; //找不到就是0

    // show content:
    for (map<char,int>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
        cout << it->first << " => " << it->second << '\n';
    return 0;
}
```

什么时侯用map？根据key快速查找value：判断key是否出现过，出现的次数，key的值？

- 不是整数和整数的对应关系。
  - 如果：给出很多学生姓名和成绩（比如有10的5次方），然后输入若干学生的姓名去查找他们的成绩（最多可能也有10的5次方）。怎么查？

用map把成绩和学生姓名map起来，这样查询的时侯就是o(logn)的时间函数了。

什么时侯用map?

- 整数和整数的对应关系，但是很大
  - 给出很多学生的编号（整数）和成绩（比如有10的5次方），然后输入若干学生的编号去查找他们的成绩（最多可能也有10的5次方）。怎么查?

如果给出的学生的编号不大，比如最大编号不超过10的4次方，显然可以用flag计数的方法，最简单，时间复杂度O(1)。数组大小差不多40k。

但是，如果编号最大会有10的9次方呢？数组大小?

用map把编号和学生成绩map起来，这样查询某个编号的成绩的时侯就是o(logn)的时间复杂度了。

什么时侯用map?

注意右面两种判断是否出现key值的方法

- 类似set，去重。
- 类似flag计数数组：查找键值去判断是否出现过或者出现了多少次。
- 下面函数判断vector数组nums中是否出现重复元素

```
bool containsDuplicate(vector<int>& nums) {
    map <int,int> mymap;
    for (int i=0;i<nums.size();i++)
    {
        if (mymap.count(nums[i])==0)
        // if (mymap[nums[i]]==0)
            mymap[nums[i]]=1;
         else return true;
    }
    return false;
}
```

map的二分查找key

给出若干个间隔（整数对），对于每一个间隔i，判断在所有间隔中是否存在另外一个间隔j，使得间隔j的开始整数大于等于间隔i的结束整数，如果有多个满足条件的j，找开始整数最小的一个间隔。

输入格式：

第一行一个整数n，表示间隔的个数，n不超过20000.

后面n行，每一个行两个整数，表示一个间隔的开始和结束整数。结束整数大于开始整数。所有间隔的开始整数都不同。

输出格式：

n行，对应输入的每一个间隔，输出题目要求的另外一个间隔的位置。（所有间隔位置从0开始）。

示例输入：

```
3
4 5
2 3
1 2
```

示例输出：

```
-1
0
1
```

第0个整数对4 5，结束是5，在所有间隔中找不到开始整数比4大的，所以输出-1

第1个整数对2 3，结束是3，显然，整数对4 5的开始是4，而整数对4 5的位置是第0个，所以输出0;

第2个整数对1 2，结束是2，显然，整数对4 5和2 3都满足条件，但是整数对2 3的开始整数是2，比4小，所以这里选择2 4这个整数对，输出1;

```cpp
#include <bits/stdc++.h>
using namespace std;
int n,a;
vector<int > rights;
map <int,int> mym;
int main(){
     cin>>n;
     for (int i=0;i<n;i++)//get input
     {
       cin>>a;
       mym[a]=i;//左边界入map作为key，区间编号i作为value
       cin>>a;
       rights.push_back(a);//右边界入数组
     }
     for (int i=0;i<rights.size();i++)
     //每一个右边界去map二分查找左边界
     {
        map <int,int>::iterator  t;
        t=mym.lower_bound(rights[i]);     //二分查找
        if (t!=mym.end())
              cout<<t->second<<endl;
        else cout<<-1<<endl;
     }
     return 0;
}
```

就是对于每一个间隔的结束整数，去看看其他的所有间隔的开始整数有没有比他大的。

最简单的思路，结束整数存一个数组，开始整数存一个数组。循环按照输入的顺序去遍历每一个结束整数：在开始整数数组中查找有没有大于等于的。

如果每一次查找都遍历数组，时间复杂度过高。

用二分法：

一个办法是输入整数的数组（需要同时保存这个整数的区间编号）排序，然后每一次都二分查找。

也可以用map来存放左边界（作为key），map会自动排序，而且map的value保存区间的编号。map里面可以用二分查找。

自己学习比较一下，map的二分和前面对于排序数组的二分查找函数有什么相同，什么不同？

下节课预习：思考右侧的题目，你有多少方法？时间复杂度都是多少？

给出一个整数序列和一个整数m，求其中和为m的连续子序列和的个数。

输入格式：

第一行2个整数n和m，n表示整数序列长度，$1=<n<=10^5$ ，m表示要找的和。$-10^5=<m<=10^5$

第二行空格分离的n个整数，每一个整数$>=-2^{31}$ 并且 $<= 2^{31} - 1$

输出格式：

一个整数，表示有多少个连续子序列的和为m。

示例输入：

5 8

-2 10 -5 3 0

示例输出：

3

作业

1. USACO 2016 December Contest, Silver

   Problem 2. Cities and States

   https://www.luogu.com.cn/problem/P3405

2. 挑战选做： USACO 2013 January Contest,

Silver Problem 3. Party Invitations

https://www.luogu.com.cn/problem/P3068

作业提示

- 作业1：
  - 也是在判断重复，如果用flag计数最简单，如何把两个字母变成数字作为flag计数的数组下标？2位的26进制数。
- 作业2：
  - 模拟，从1号开始，不停的从其他奶牛的朋友圈里面把找到的删除，什么时侯某一个奶牛朋友圈剩下一个了，这个奶牛也要选。
  - 最关键是用什么数据结构。可以用set数组，每一个set元素来保存当前一组中的奶牛编号。然后从1号奶牛开始处理：每一次遍历所有组的set，把组里面该元素删除，然后如果当前某一个set只有一个元素了，就取出来进行下一次操作。

由易到难，思维体系训练
实战结合，创新协作培养
兴趣导向，未来职业引领

https://www.35tang.com

https://www.三五堂.com

扫码关注公众号

添加辅导老师