ACM 算法与微应用开发实验室 21 届成员选拔赛题解

2021 年 11 月 6 日

题目概览

题目编号	题目名称	命题人	做法
A	??	??	线性筛
B	??	??	搜索
C	??	??	贪心
D	??	??	前缀和
E	??	??	模拟
F	??	??	模拟

注意：因篇幅显示，部分标程省略了头文件部分。

本次比赛在 Github 上开源 (https://github.com/jgsu-acm/jgsu-acmlab-trial-2021)。包括通知、题目、题解的 LaTeX 源码及所有标程。

鸣谢

感谢 Tifa 大佬为本次比赛贡献的题目。

A. ??

做法

本题考查对线性筛算法的理解。

我们回想一下线性筛在干什么，首先我们有一个 visit 数组用来存我们是否已经筛过当前的数，还有个 prime 数组记录我们筛出的素数。我们从 2 开始遍历，如果当前的数没被筛掉，那它就是素数。同时对于我们枚举的每个数 i，把 i 与素数 p_j 的乘积筛掉，也就是在 visit 数组打上标记。p_j 要满足 $p_j < p$, 其中 p 是 i 的最小素因子，这样能确保每个数都会被不重不漏地访问一遍。

那我们怎么找 $k-$ 素数呢？

我们再定义一个数组 k_cnt[], k_cnt[i] 表示 i 能表示为多少个素数的乘积。显然，对所有的素数 p, k_cnt[p] 都应赋为 1。并且 k_cnt[i * p] = k_cnt[i] + 1。

所以我们只需要在标记 $i \times p_j$ 时把 k_cnt[i * prime[j]] 赋为 k_cnt[i] + 1 即可。

完全 $k-$ 素数同理。

标程

```
#include <iostream>
using namespace std;
const int maxn = 2e6+5;
bool vis[maxn];
int cnt, prime[maxn];
int kcnt[maxn], exkcnt[maxn];
int kcntmax[maxn], exkcntmax[maxn];
int main()
{
    // 首先进行线性筛
    for(int i=2;i<maxn;i++)
    {
        if(!vis[i])
        {
            prime[cnt++]=i;
```

```
            kcnt[i]=exkcnt[i]=1;
        }
        for(int j=0;j<cnt&&i*prime[j]<maxn;j++)
        {
            vis[i*prime[j]]=true;   // 这个数不是质数
            kcnt[i*prime[j]]=kcnt[i]+1; // 显然i*prime[j]这个数因为多乘上了一个质
                数所以可以由kcnt[i]+1个质数相乘来得到
            if(i%prime[j]==0) break;    // 找到了i的最小素因子
            // 当i和prime[j]互质时继续执行
            if(exkcnt[i]) exkcnt[i*prime[j]]=exkcnt[i]+1;   // 只有当i没有平方因子
                的时候才是完全k素数
        }
    }
```

```
for(int i=2;i<maxn;i++) // 处理出前i个数中最大由多少个质数相乘得到
{
    kcntmax[i]=max(kcntmax[i-1], kcnt[i]);
    exkcntmax[i]=max(exkcntmax[i-1], exkcnt[i]);
}
int t;
cin>>t;
while(t--)
{
    int n,k;
    cin>>n>>k;
    if(k==0)    // 能由0个质数乘出来的数字只有1
    {
        cout<<"1 1"<<endl<<"1 1"<<endl;
        continue;
    }
    int ktot=0, kans=0, exktot=0, exkans=0; // 注意异或和初值为0因为0异或任何
        数结果等于这个数
    if(k<=kcntmax[n])   // 质数因子个数最多的数字都不能由k个质数拼出来的话直接
        byebye。以下同理
    {
        for(int i=1;i<=n;i++)
            if(kcnt[i]==k)
                kans ^= i, ktot++;
    }
    if(k<=exkcntmax[n])
    {
        for(int i=1;i<=n;i++)
            if(exkcnt[i]==k)
                exkans ^= i, exktot++;
    }
    cout<<ktot;
    if(ktot) cout<<' '<<kans; cout<<endl;
    cout<<exktot;
    if(exktot) cout<<' '<<exkans; cout<<endl;
}
return 0;
```

```
}
```

B. ??

做法

数据规模较小，直接根据题意搜索模拟马跳的过程即可。

标程

```
#include <iostream>
#include <cstring>
using namespace std;
int ans[15][15];
int x2,y2;  // 终点
void dfs(int x,int y,int cnt)   // x,y：当前走到了哪个点；cnt：当前已走了几步
{
    // 其实还可以进行其它剪枝，留作思考题
    if(cnt>=ans[x][y]) return;   // 若当前花费的步数已经大于等于之前走到此处花费的
        步数，则不用继续走了
    ans[x][y]=min(ans[x][y],cnt);   // 更新答案
    if(x==x2&&y==y2) return;    // 若走到终点则不用走了
    // 分别尝试往八个方向走，并且使花费的步数+1。注意用if判断防止走到棋盘外
    if(x-2>=1&&y-1>=1) dfs(x-2,y-1,cnt+1);
    if(x-2>=1&&y+1<=9) dfs(x-2,y+1,cnt+1);
    if(x-1>=1&&y-2>=1) dfs(x-1,y-2,cnt+1);
    if(x-1>=1&&y+2<=9) dfs(x-1,y+2,cnt+1);
    if(x+1<=10&&y-2>=1) dfs(x+1,y-2,cnt+1);
    if(x+1<=10&&y+2<=9) dfs(x+1,y+2,cnt+1);
    if(x+2<=10&&y-1>=1) dfs(x+2,y-1,cnt+1);
    if(x+2<=10&&y+1<=9) dfs(x+2,y+1,cnt+1);
}
int main()
{
    int x1,y1;
```

```
    cin>>x1>>y1>>x2>>y2;
    memset(ans, 0x3f, sizeof(ans));     // 要求最小值，ans数组设置为无穷大
    dfs(x1,y1,0);
    cout<<ans[x2][y2]<<endl;
    return 0;
}
```

C. ??

做法

因为只有弹出操作，所以本题并不是考察双端队列的用法，只是起了这样一个名字而已……

例如对于题目样例给出的双端队列 $2,1,5,4,3$，第一次弹出应该弹出什么？显然应该弹出 2 而不是 3，因为若弹出 3 则再也不能弹出 2 了，但弹出 2 后还可以弹出 3。

所以本题利用贪心思想即可解决：在两侧都能弹出的时候弹出数字较小的一侧的数字；否则哪边能弹出就弹出哪边；当两边都不能弹出或队列为空时输出答案即可。

因为题目所给数据只是 $1 \sim n$ 的一个排列（数字不会重复），所以难度较低。本题也可以扩展到数字可以重复的情况，留作思考题。

标程

```
#include <iostream>
using namespace std;
const int maxn = 5e5+5;
int q[maxn];
int main()
{
    int n;
    cin>>n;
    for(int i=0;i<n;i++) cin>>q[i];
    int l=0, r=n-1, cnt=0, last=-1;         // last用于记录上一次弹出的数据，初始
        为无穷小
    while(l<=r)
    {
        if(q[l]<last&&q[r]<last) break;
        ++cnt;
        if(q[l]>last&&q[r]>last)            // 若两边都能弹出，弹出数值较小的一边
```

```
        {
            if(q[l]<q[r]) last=q[l++];
            else last=q[r--];
        }
        else if(q[l]>last) last=q[l++];     // 只有左边能弹出
        else last=q[r--];                   // 只有右边能弹出
    }
    cout<<cnt<<endl;
    return 0;
}
```

D. ??

做法

暴力做法 对于每次询问都使用循环来统计区间内数字和的方法的时间复杂度为 $O(mn)$，对于本题 $1e5$ 的数据规模显然不能通过。

正解 需要使用一种叫做“前缀和”的简单算法。

注意到

$$a_l + a_{l+1} + \cdots + a_r = (a_1 + a_2 + \cdots + a_r) - (a_1 + a_2 + \cdots + a_{l-1})$$

我们定义

$$pre_i = a_1 + a_2 + \cdots + a_i$$

于是

$$a_l + a_{l+1} + \cdots + a_r = pre_r - pre_{l-1}$$

所以对于每次 $[l,r]$ 查询，只需要 $O(1)$ 的时间复杂度即可得出结果，而 pre_i 可以通过 $pre_i = pre_{i-1} + a_i$ 的递推式从而在 $O(n)$ 的时间复杂度内推出来。总时间复杂度为 $O(n+m)$。

标程

```
#include <iostream>
using namespace std;
const int maxn = 1e5+5;
int pre[maxn];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)       // O(n)处理出pre (pre[i]=pre[i-1]+a[i])
    {
```

```
        int x;
        cin>>x;
        pre[i]=pre[i-1]+x;
    }
    while(m--)                  // O(m)处理所有询问
    {
        int l,r;
        cin>>l>>r;
        cout<<pre[r]-pre[l-1]<<endl;
    }
}
```

E. ??

做法

一个模拟题，写法很多。这里给出其中一种

我们考虑枚举出雀头，删去雀头的两张牌后判断剩余的 12 张牌是否为 4 个面子。

我们可以直接对每种牌模 3，之后判断剩下的牌是不是顺子即可

标程

```
#include <iostream>
#include <cstring>
using namespace std;
int mj[5][15], cpy[5][15];  // mj一维从0到3分别代表万饼条字牌，二维代表点数
bool check2()
{
    memcpy(cpy, mj, sizeof(mj));            // 此函数用途为往cpy里复制一个mj，防止
        误伤到mj数组
    for(int i=1;i<=9;i++)
    {
        for(int j=0;j<4;j++) cpy[j][i]%=3;  // 删掉所有的刻子
        if(cpy[3][i]) return false;         // 因为字牌不能成顺子，所以若还剩下了
            字牌则必然不能和牌
        // 因为雀头和刻子都没了所以以下只需要判断顺子
        for(int j=0;j<3;j++)                // 遍历顺子的最小牌
        {
            if(cpy[j][i])
            {
                if(i>7) return false;       // 如果还有多余的8,9则必然不能成顺子
                // 删除顺子
                if(cpy[j][i+1]<cpy[j][i]||cpy[j][i+2]<cpy[j][i])    // 如果不能成
                    顺子
```

```
                    return false;
                cpy[j][i+1] -= cpy[j][i];
                cpy[j][i+2] -= cpy[j][i];
                // 因为cpy[j][i]以后不会被访问到，所以cpy[j][i]这个位置可以不减
            }
        }
    }
    return true;
}
```

```
bool check1()
{
    bool flag;
    for(int i=1;i<=9;i++)
    {
        for(int j=0;j<4;j++)
        {
            if(mj[j][i]>1)          // 枚举雀头
            {
                mj[j][i]-=2;        // 删掉雀头
                flag = check2();    // 判断
                mj[j][i]+=2;        // 再把雀头加回来
                if(flag) return true;
            }
        }
    }
    return false;
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        memset(mj, 0, sizeof(mj));
        for(int i=0;i<14;i++)
        {
            string s;
            cin>>s;
            int x = s[0]-'0';
            switch(s[1])
            {
            case 'w': mj[0][x]++; break;
            case 'b': mj[1][x]++; break;
            case 't': mj[2][x]++; break;
            case 'z': mj[3][x]++; break;
            }
```

```
        }
        cout << (check1() ? "Tsumo!" : "Waiting for Tsumo!") << endl;
    }
    return 0;
}
```

F. ??

做法

签到题，题意即求出 $\lfloor A\times(1-p1\%)\times(85\%)^0\rfloor+\lfloor A\times(1-p2\%)\times(85\%)^1\rfloor+\lfloor A\times(1-p3\%)\times(85\%)^2\rfloor+\lfloor A\times(1-p4\%)\times(85\%)^3\rfloor$ 的值。可以使用秦九韶算法，也可以cout一行搞定。

标程

cout一行搞定：

```
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int A,p1,p2,p3,p4;
        cin>>A>>p1>>p2>>p3>>p4;
        cout<<((int)(A*(1-p1/100.0)) + (int)(A * 0.85 * (1-p2/100.0)) + (int)(A *
            0.85 * 0.85 * (1-p3/100.0)) + (int)(A * 0.85 * 0.85 * 0.85 * (1-p4
            /100.0)))<<endl;
    }
    return 0;
}
```

秦九韶算法：

```
int p[7];
int main()
{
    int t;
    cin>>t;
    while(t--)
```

```
    {
        int A;
        cin>>A;
        for(int i=1;i<=4;i++) cin>>p[i];
        double mul = 1;
        int sum = 0;
        for(int i=1;i<=4;i++, mul *= 0.85)
            sum += (int)(A * mul * (1 - p[i] / 100.0));
        cout<<sum<<endl;
    }
    return 0;
}
```