上海理工校赛题解报告

小结

数学题，不会

B题 Bheith i ngra le

当初挣扎了一下，然后不会求给定i行j列的格子有多少单调曲线，就放弃了。

分析

- 题目的核心是是求给定i行j列的格子能够构造多少调单调曲线。可以用dp或者组合求解，不喜欢数学的我毅然投奔了dp。
- 不难发现，当给定了宽为i，高为j的格子矩阵的时候，我们分析右上角那块，如果这块不画，那么所有情况就会变成dp[i][j-1],如果这块画上，那么情况就会是dp[i-1][j]。则得到状态转移方程dp[i][j]=dp[i][j-1]+dp[i-1][j]。当然，边界条件还是值得考虑一下的。
- 算出dp数组后，性高彩烈的用n^3去套老鹅。其实我们确定了山顶左边位置l之后不用去确定右边的r，dp[n-l][h]就是右边包含山顶在内的全部可能。所以ans=ans+dp[l-1][h-1]*dp[n-l][h]即可。

```
#include<iostream>
using namespace std;
long long dp[2003][2003];
long long mod=1e9+7;
long long ans;
int main(){
    int n,m;
    cin >>n>>m;
    // 初始化高度为1时
    dp[1][1]=2;
    for(int i = 2;i<=n;++i){
        dp[i][1]=dp[i-1][1]+1;
    }
    // 初始化宽度为1的时候
    for(int j = 2;j<=m;++j){
        dp[1][j]=dp[1][j-1]+1;
    }
    // 求dp，i为宽，j为高
    for(int i = 2;i<=n;++i){
        for(int j = 2;j<=m;++j){
           dp[i][j]=(dp[i][j-1]+dp[i-1][j])%mod;
        }
    }

    // 宽度为0时答案应该为1，因为下面用的乘法
    for(int j=0;j<=m;++j){
        dp[0][j]=1;
    }

   // 高度为0时，也应该是1，即取0
   for(int i = 0;i<=n;++i){
       dp[i][0]=1;
```

```
    }

    // cout<<dp[1][1]<<endl<<dp[2][1]<<endl<<dp[1][2]<<endl<<dp[2][2]<<endl<<dp[1][3]
<<endl<<dp[2][3]<<endl;

    // 枚举山顶的情况,l是左边，r是右边，h是山高
    for(int l = 1;l<=n;++l){
            for(int h = 1;h<=m;++h){
               ans = (ans+(dp[l-1][h-1]*dp[n-l][h])%mod)%mod;    // 是左边的情况乘以右边
的情况
            }
    }

    // 加上山顶全为0的情况，只有一种
     ans=(ans+1)%mod;

    cout<<ans<<endl;
}
```

J题 JXC&JESUS

这是一个数学题

分析

题目大概的意思是定义了一个函数

$$f(i,m)=p_1^{\lfloor\frac{a1}{m}\rfloor}*p_2^{a_2}*\ldots*p_{a^k}^k$$

其中p_1,p_2 ...p_k为i的质因数。给定参数n,m,L求

$$\sum_{i=L+1}^{L+n}-f(i,m)$$

这关系到i的质因数的问题，可以用线性筛求质因数。又f(i,m)只和m与其最小质因子有关，我们在线性筛的时候合数也是被其最小质因子筛掉。所以想到从线性筛的基础上进行状态转移，由小的质数推出合数的f(i,m)，从而使求答案也变成线性。另一种是求出了素数之后直接暴力，也能过。
根据以下算法枚举 i 进行计算

- 当i是质数的时候，当m=1时，f[i]=i,否则f[i]=1。（注意：1不是质因数）
- 枚举质数prime[j]利用i对i*prime[j]进行推算。
    - 当prime[j]不能整除i时，则prime[j]是$i*prime[j]$的最小质因子，且只有一个，$cnt[i*prime[j]]=1$。当m = 1时 ,$f[i*prime[j]]=i*prime[j]$,否则$f[i*prime[j]]=i$。(因为此时$\lfloor\frac{a1}{m}\rfloor$为0)
    - 当prime[j]能整除i时，则prime[j]也是i的最小质因子，此时$cnt[i*prime[j]]=cnt[i]+1$。当$cnt[i*prime[j]]$ 时，$f[i*prime[j]]=i*prime[j]$,否则$f[i*prime[j]]=i$。（此时应该枚举新的i，就如线性筛那样。）

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

```
#define MAX 20000003
int prime[MAX];   // 存素数
bool isprime[MAX]; // 判断是否素数
int n,m,L;
int ans=0;   // 存放答案
int cnt[MAX]; // 存放最小质因数的个数
int CNT=1;
int f[MAX];  // f存起来，因为可能提前求得
int minp[MAX]; // 记录合数的最小质因子

// 求素数
void calPrime(){
    prime[0]=1;
    isprime[0]=isprime[1]=isprime[2]=true;
    for(int i = 2;i<=MAX;++i){
        if(isprime[i]){
            prime[CNT++]=i;
        }
        // 将i的倍数标记为合数，合数只会被其最小质因子标记，所以很方便可求出最小质因子
        for(int j = 1;j<CNT&&i*prime[j]<=MAX;++j){
            isprime[i*prime[j]]=false;
            // 如果i能够整除prime[j]，则退出
            if(i%prime[j]==0){
                break;
            }
        }
    }
}
// 直接求f
void  F(){
    for(int i =2;i<=L+n;++i){

        // i是质数的情况,其最小质因数为其本身,则根据m讨论f
        if(isprime[i]){
            if(m==1){
                f[i]=i;
                cnt[i]=1;
            }else{
                f[i]=1;
                cnt[i]=1;
            }
        }
        // i不是质数的情况,枚举质数，不包括1
        for(int j = 1;j<CNT&&i*prime[j]<=L+n;++j){
            // 如果i不能整除prime[j],此时prime[j]是i*prime[j]的最小质因子，因为i非质数
           if(i%prime[j]!=0){
               if(m==1){
                   f[i*prime[j]]=i*prime[j];
               }else{   //m!=1时，最最小质因数就无贡献了
                   f[i*prime[j]]=i;
               }
               cnt[i*prime[j]]=1;
           }else{ // 如果prime[j]能整除当前的i，则prime[j]也是i的最小质因子
               cnt[i*prime[j]]=cnt[i]+1;
               // 只有当cnt是m的倍数的时候，prime[j]才有贡献
               if(cnt[i*prime[j]]%m==0){
                   f[i*prime[j]]=f[i]*prime[j];
```

```
                }else{
                    f[i*prime[j]]=f[i];
                }
                break;

            }
        }
    }
}

int main(){
    memset(isprime,true,sizeof(isprime));
    memset(cnt,1,sizeof(cnt));
    memset(f,0,sizeof(f));
    cin>>n>>m>>L;
    calPrime(); //求素数
    F();
    for(int i = L+1;i<=L+n;++i){
        ans+=i-f[i];
    }
    cout<<ans<<endl;
}
```

M题 Minecraft

这题挺简单，毕竟人家说了时签到题，不过这个输入格式确实有点费脑。

分析

根据题目，差不多重点就是下面

- 每次从房子的每根"柱子"最底下建起方块(因为不能流沙)。
- 每次选定一个字母之后所以该字母的位置都要建方块！
- 最后要以最大字典序输出！

可以通过有向图和拓扑排序解决，先填充入度为0的字母，然后更新图，如果全部入度为0的都已经访问了还有入度不为0，就重建失败。
对于字典序问题，没有用优先队列，直接在while里面套一个for即可，并且要从字母Z遍历到A，特别注意的是，一旦找到一个字母后，要重新进行for循环，不然达不到字典序。就因为这个卡88.9%给我整吐了。

```
#include<iostream>
#include<cstring>
using namespace std;
#define MAX 27
int t;   // 测试数
int n,m,h;  // 长宽高
int let[35][35][35];   // 在房子某个坐标的字母,用int存
char c;
typedef struct NODE{
    int in=0;   // 入度
    int out=0;  // 出度
    int next[30000];  // 连接的下一个点，注意点的个数
    bool isused=false;  // 记录是否已经访问过
```

```
    bool isExit=false;
}NODE;
char ans[MAX]; // 存放答案的字母
int  cnt=0;   // 答案中共有多少个字母
NODE Node[MAX];    // 设置结点
int main(){
    //freopen("test/USSTM.txt","r",stdin);
    cin>>t;
    while(t--){
        // 每次开始的时候记得还原

        cnt=0;
        c=0;
        for(int i = 0;i<MAX;++i){
            Node[i].in=0;
            Node[i].isExit=false;
            Node[i].isused=false;
            Node[i].out=0;
        }
        cin>>n>>m>>h;
        if(n==0||m==0||h==0){
            cout<<"-1"<<endl;
            continue;
        }

       // 输入考了一波理解
        for(int k = h-1;k>=0;--k){
        for(int i = 0;i<n;++i){
            for(int j = 0;j<m;++j){

                cin>>c;
               let[i][j][k]=(int)c-'A';
               Node[let[i][j][k]].isExit=true;

            }
         }

        }


        // 开始构造图
        for(int k = 1;k<h;++k){
        for(int i = 0;i<n;++i){
            for(int j = 0;j<m;++j){
                // 如果和下面的不相等，则要生成边
                if(let[i][j][k]!=let[i][j][k-1]){
                    int tempU=let[i][j][k];
                    int tempL=let[i][j][k-1];
                    Node[tempU].in++;
                    Node[tempL].next[Node[tempL].out++]=tempU;
                }
            }
        }
    }

        bool canUsed=true;    // 记录能否找到入度为0且没有访问过的结点
```

```
        while(canUsed){
        canUsed=false;
        // 从字母大的找起,这里就不用优先队列了
        for(int i = 25;i>=0;--i){
           // 本来有这个点才去判断
           if(Node[i].isExit){
                // 入度为0且没有访问过
                if(Node[i].in==0&&!Node[i].isused){
                    canUsed=true;
                    ans[cnt++]=char(i+'A');
                    // 更新结点状态,并更新其连接的边的状态
                     Node[i].isused=true;
                    for(int j = 0;j<Node[i].out;++j){
                         Node[Node[i].next[j]].in--;
                    }

                    // 找到一个之后，一定要重新再来，不然达不到字典序
                    break;
                }
           }
        }

    }

        // 判断是否还有没有访问过的本来存在的点
        bool flag=false;
        for(int i = 25;i>=0;--i){
           // 本来有这个点才去判断
           if(Node[i].isExit){
                if(!Node[i].isused){
                    flag =true;
                    break;
                }
           }
        }

        if(flag){
            cout<<"-1"<<endl;
        }else{
            for(int i = 0;i<cnt;++i){
                cout<<ans[i];
            }
            cout<<endl;
        }

    }

}
```