2022/11/14 总结

T1 substr

题意

- 给出 n 个字符串 S_i
- 求一个最短的字符串 T，使得 S_i 都是 T 的字串
- 如果有多种情况，输出字典序最小的 T
- $1 \leq n \leq 12, 1 \leq |S_i| \leq 50$

分析

- 首先，很明显的，要使 T 最短，就是使 S_i 的后缀尽可能地与 S_{i+1} 的前缀匹配
- 于是考虑 AC 自动机。
  - 对每个节点用 st 表示这个点是哪个字符串的结尾（状压）
  - 构建 $fail$ 数组的同时，如果节点 p 有 $son_{p,c}$，则在 $fail_{son_{p,c}} = son_{fail_p,c}$ 后，使 $st_{son_{p,c}} = st_{son_{p,c}} \vee st_{fail_{son_{p,c}}}$（将这个节点的 $fail$ 指向的节点的信息归到这个节点上）
  - 建完后，直接跑一边 bfs（最短路）就行，过程中记录当前搜到的点的位置 x、代表的字符 c、st（判断是否已包含所有 S_i）、pre（从哪个点到达这个点，方便最后输出）

代码

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

const int M=1<<12;
int n,len,tot,son[610][26],fail[610],st[610],num,head,tail;
char s[55],ans[610];
bool vis[610][M];

void insert()                //字典树插入字符串
{
    int p=0;
    for(int i=1;i<=len;++i)
    {
        if(!son[p][s[i]-'A']) son[p][s[i]-'A']=++tot;
        p=son[p][s[i]-'A'];
    }
    st[p]|=1<<num;
    ++num;
}

void build()                    //构建fail，同时维护节点st
{
    queue<int>que;
    for(int i=0;i<26;++i)
        if(son[0][i])
            que.push(son[0][i]);
    int p;
```

```
    while(!que.empty())
    {
        p=que.front();
        que.pop();
        for(int i=0;i<26;++i)
        {
            if(son[p][i])
            {
                fail[son[p][i]]=son[fail[p]][i];
                st[son[p][i]]|=st[fail[son[p][i]]];
                que.push(son[p][i]);
            }
            else son[p][i]=son[fail[p]][i];
        }
    }
}

struct node
{
    int x,c,st,pre;
}que[610*M];

void slove()                        //bfs
{
    que[++tail]=(node){0,0,0,0};
    node now,nex,nxt;
    while(head!=tail)
    {
        now=que[++head];
        if(now.st==(1<<n)-1)
        {
            int cnt=0;
            for(int i=head;i;i=que[i].pre)
                ans[++cnt]=que[i].c+'A';
            for(int i=cnt-1;i;--i)
                putchar(ans[i]);
            return;
        }
        for(int i=0;i<26;++i)
        {
            nex.x=son[now.x][i],nex.c=i,nex.st=now.st|st[nex.x],nex.pre=head;
            if(!vis[nex.x][nex.st])
            {
                vis[nex.x][nex.st]=1;
                que[++tail]=nex;
            }
        }
    }
}

int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i)
        scanf("%s",s+1),len=strlen(s+1),insert();
```

```
    build();
    slove();
    return 0;
}
```

T2 flower

题意

- 给定 n 个值 a_i
- 给出 m 个询问，每次给出 l_i, r_i, k_i，输出 $\max_{j=l_i}^{r_i}(a_j \mod k_i)$
- $1 \le n, m, a_i, k_i \le 10^5$

分析

- 首先，$O(nm)$ 的暴力十分显然，对于每个询问，直接枚举 $a_{l_i \sim r_i}$，取最大值就行
- 那要如何优化呢？注意到一个性质，对于一个区间 $l \sim r$，设 R 表示值域，A_i 表示 $\max_{j=1}^{i} R_j$，则他们取模 k 的最大值既是 $\max\limits_{k|i} A_i - i + k$，于是我们就可以在 $O(N + \frac{N}{k})$ 的时间复杂度内求出任意一个区间对任意数取模的最大值
- $O(N)$ 是用在将数映射到值域上的时间，当一个区间对多个数取模时，只需要映射一次，所以我们可以在 $O(N \cdot \sum_{k=1}^{N} \frac{N}{k})$，即 $O(N \ln N)$ 的时间复杂度内处理出一个区间对所有数取模的最大值
- 于是就可以将序列分块，设块的大小为 B，就可以在 $O(\frac{n}{B} N \ln N)$ 的时间内预处理出所有块对所有数取模的最大值，再以 $O(B + \frac{n}{B})$ 的时间回答每个询问，总时间复杂度 $O(\frac{n}{B} N \ln N + mB + m\frac{n}{B})$

代码

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

template<typename T>inline void re(T &x)
{
    T f=1;x=0;
    char c=getchar();
    while(!isdigit(c)){if(c=='-')f=-1;c=getchar();}
    while(isdigit(c))x=(x<<3)+(x<<1)+(c^48),c=getchar();
    x*=f;
}

const int N=1e5+10;
int n,m,a[N],ans[N][110],mx[N],Bsize,Bnum;

void init_()
{
    Bsize=1000;
    Bnum=(n-1)/Bsize+1;
    for(int k=1;k<=Bnum;++k)
    {
        memset(mx,0,sizeof(mx));
        int up=min(Bsize*k,n);
```

```
        for(int i=Bsize*(k-1)+1;i<=up;++i)
            mx[a[i]]=a[i];
        for(int i=1;i<N;++i)
            mx[i]=max(mx[i],mx[i-1]);
        for(int i=1;i<N;++i)
            for(int j=0;j<N;j+=i)
                ans[i][k]=max(ans[i][k],mx[min(j+i-1,N-1)]-j);
    }
}

int main()
{
    re(n),re(m);
    for(int i=1;i<=n;++i)
        re(a[i]);
    init_();
    int l,r,k,lb,rb,res;
    while(m--)
    {
        re(l),re(r),re(k);
        res=0;
        lb=(l-1)/Bsize+1,rb=(r-1)/Bsize+1;
        if(lb==rb)                              //特判在一个块内的情况
        {
            for(int i=l;i<=r;++i)
                res=max(res,a[i]%k);
        }
        else
        {
            for(int i=lb+1;i<rb;++i)
                res=max(res,ans[k][i]);
            for(int i=l;i<=lb*Bsize;++i)
                res=max(res,a[i]%k);
            for(int i=(rb-1)*Bsize+1;i<=r;++i)
                res=max(res,a[i]%k);
        }
        printf("%d\n",res);
    }
    return 0;
}
```

T3 refract

题意

- 给出 n 个点的坐标 x_i, y_i
- 一条合法的路径 $(x_1,y_1),(x_2,y_2),(x_3,y_3)\cdots(x_k,y_k)$，满足 $\forall i\in(1,k], y_i<y_{i-1}$ 且 $\forall i\in(2,k], x_{i-2}<x_i<x_{i-1}\vee x_{i-1}<x_i<x_{i-2}$
- 求出所有合法路径的总数对 10^9+7 取模的结果（单个点也算合法路径）
- $1\le n\le 6000$，$|x_i|,|y_i|\le 10^9$，$\forall i\neq j, x_i\neq x_j, y_i\neq y_j$

分析

- 很容易想到，将所有点按 y_i 降序排序，再去满足 x_i 的条件
  - 设 $dp_{i,j}$ 表示到第 i 个点结束，上一个点的横坐标坐标为 j 的方案数
  - 则有转移方程

$$dp_{i,j}=\sum_{x_k=j,k<i}\sum_{j'>x_i}dp_{k,j'}+\sum_{x_k=j,k<i}\sum_{j'<x_i}dp_{k,j'}$$

  - 直接转移是 $O(n^4)$ 的，枚举 i,k 是 $O(n^3)$，再随便用个前缀和优化就是 $O(n^2)$ 的
  - 但是很遗憾，他的空间复杂度也是 $O(n^2)$ 的，在题目的限制下过不了
- 于是考虑将所有点按 x_i 升序排序
  - 设 $dp_{i,0/1}$ 表示从第 i 个点出发，接下来向左/右拐的路径数
  - 从小到大枚举 i，再从 i 向前枚举另一个点 j，如果 $y_j<y_i$，则有

$$dp_{i,0}=dp_{i,0}+dp_{j,1}$$

    如果 $y_j>y_i$，因为之前枚举的 j 都在这个 j 的右边，所以则有

$$dp_{j,1}=dp_{j,1}+dp_{i,0}$$

  - 于是可以在 $O(n^2)$ 的时间复杂度，$O(n)$ 的空间复杂度内通过

代码

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

const int N=6010,P=1000000007;
int n,ans,dp[N][2];

struct node
{
    int x,y;
}a[N];

bool cmp(node x,node y)
{
    return x.x<y.x;
}

int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i)
        scanf("%d%d",&a[i].x,&a[i].y);
    sort(a+1,a+n+1,cmp);
    for(int i=1;i<=n;++i)
    {
        dp[i][0]=dp[i][1]=1;
        for(int j=i-1;j;--j)
        {
            if(a[i].y>a[j].y)
```

```
                dp[i][0]+=dp[j][1],dp[i][0]%=P;
            else
                dp[j][1]+=dp[i][0],dp[j][1]%=P;
        }
    }
    for(int i=1;i<=n;++i)
        ans=(ans+dp[i][0])%P+dp[i][1],ans%=P;
    printf("%d",(ans-n+P)%P);
    return 0;
}
```

T4 paint

题意

- 给定一个 $n \times m$ 的 $0,1$ 目标矩阵
- 一次操作可以使一个连通块都变为 $0/1$
- 求经过最少多少次操作，可以使全 0 的矩阵变为目标矩阵
- $1 \leq n,m \leq 50$

分析

- 有这样一个结论：对于每一次操作，他改变的一定是上一次操作的子集，且颜色相反

> 设 T 为当前操作的集合，S 为上一次操作的集合
>
> 1. T 与 S 有交的情况一定可以转化成 T 被 S 包含的情况
> 2. T 与 S 交集为空时，可以找一个连接 S 和 T 的集合 M 并操作 $S \cup M \cup T$，并将之前的所有操作连接到更外的层以及外层的连接部分同时操作, 特殊处理最外层和第二层的情况
> 3. T 被 S 包含时，T 落在某个完整区域内时等价于情况二，否则一定连接若干个同色块，这些块可以同时处理，步数一定不会更劣

- 于是我们可以枚举最后操作的格子，以他为起点跑最短路（$0/1$ BFS），如果两个格子颜色相同，则边权为 0，否则为 1，距离最远的格子的值就是操作次数，最后取最小就行。注意当操作次数为奇数，所枚举格子为 0，和操作次数为偶数，格子为 1 时要多操作一次
- 时间复杂度 $O(n^2m^2)$

代码

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

const int N=55,dx[4]={0,1,0,-1},dy[4]={1,0,-1,0};
int n,m,ans=1e9;
bool vis[N][N];
char a[N][N];

struct node
{
    int x,y,d;
};
```

```
deque<node>que;

bool pd(int x,int y)                    //判断越界和是否已搜过
{
    if(x<1||x>n||y<1||y>m) return 0;
    return 1;
}

void bfs(int x,int y)                   //0/1bfs
{
    int tmp=0;
    memset(vis,0,sizeof(vis));
    que.push_back((node){x,y,0});
    node now,nex;
    while(!que.empty())
    {
        now=que.front();
        que.pop_front();
        if(vis[now.x][now.y])
            continue;
        vis[now.x][now.y]=1;
        tmp=max(tmp,now.d);
        for(int i=0;i<4;++i)
        {
            nex.x=now.x+dx[i],nex.y=now.y+dy[i];
            if(!pd(nex.x,nex.y)) continue;
            nex.d=now.d+(a[now.x][now.y]!=a[nex.x][nex.y]);
            if(a[now.x][now.y]==a[nex.x][nex.y])
                que.push_front(nex);
            else
                que.push_back(nex);
        }
    }
    ans=min(ans,tmp+((a[x][y]^48)^(tmp&1)));
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;++i)
        scanf("%s",a[i]+1);
    for(int i=1;i<=n;++i)
        for(int j=1;j<=m;++j)
            bfs(i,j);
    printf("%d",ans);
    return 0;
}
```