2022/11/12 总结

T2 dealing

题意

- 给定字符串长度 n 和操作次数 m
- 给出 m 个操作 x_i, y_i, len_i，表示将 $x_i \sim x_i + len_i - 1$ 与 $y_i \sim y_i + len_i - 1$ 的对应位置的字符规定为相同
- 求不同的字符串个数（仅包含小写英文字母）
- $1 \le n \le 10^6, 0 \le m \le 2 \times 10^5, 1 \le len_i \le n, 1 \le x_i, y_i \le n - len_i + 1$

分析

- 首先，题目就是让我们求最后可以任意取值的位置 cnt，输出 26^{cnt}
- 于是有个很明显的暴力思路。用并查集维护同一取值的块，每次操作既是将 $x_i \sim x_i + len_i - 1$ 的每一位与 $y_i \sim y_i + len_i - 1$ 的每一位一一合并，最后统计连通块的个数既是 cnt。时间复杂度 $O(\sum len_i)$
- 那要如何优化合并的过程呢？我们发现每次合并的都是一段连续的区间，于是就可以将区间合并，而不是每个点
  - 设 $F_{i,j}$ 表示 $i \sim i + 2^j - 1$ 的连通块的根（就是并查集的 fa 数组）
  - 每次操作时，将 len_i 二进制拆分，将区间上的每一部分合并。具体地说，设 len_i 的第 k 位是 1，则将 $F_{x_i,k}$ 和 $F_{y_i,k}$ 合并，同时将 x_i 加上 2^k，y_i 加上 2^k
  - 最后统计答案之前，先将 $F_{i,j}$ 下放到 $F_{i',j-1}$。具体地说，从高到低枚举每一位 j，将 $F_{i,j-1}$ 和 $F_{F_{i,j},j-1}$ 合并，再将 $F_{i+2^{j-1},j-1}$ 和 $F_{F_{i,j}+2^{j-1},j-1}$ 合并
  - 最后一样用 $F_{i,0}$ 统计连通块个数

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=1e6+10,P=1000000007;
int n,m,F[N][21],ans;
bool vis[N];

int find(int x,int k)                          //并查集查找
{
    if(F[x][k]==x) return x;
    return F[x][k]=find(F[x][k],k);
```

```
}

inline void merge(int x,int y,int k)        //并查集合并
{
    if(x>n||y>n) return;
    if(find(x,k)!=find(y,k))
        F[find(x,k)][k]=find(y,k);
}

inline int mpow(int x,int y)                //快速幂
{
    int res=1;
    for(;y;y>>=1)
    {
        if(y&1) res=1ll*res*x%P;
        x=1ll*x*x%P;
    }
    return res;
}

int main()
{
    re(n),re(m);
    for(int i=1;i<=n;++i)
        for(int j=0;j<=20;++j)
            F[i][j]=i;
    int l,x,y;
    while(m--)
    {
        re(l),re(x),re(y);
        for(int i=0;i<=20;++i)              //处理每个操作
            if(l>>i&1)
            {
                merge(x,y,i);
                x+=1<<i;
                y+=1<<i;
            }
    }
    for(int i=20;i;--i)                     //下放
    {
        for(int j=1;j<=n;++j)
        {
            merge(j,find(j,i),i-1);
            merge(j+(1<<i-1),find(j,i)+(1<<i-1),i-1);
        }
    }
    for(int i=1;i<=n;++i)                   //统计连通块个数
        if(!vis[find(i,0)])
        {
            vis[find(i,0)]=1;
            ++ans;
        }
    printf("%d",mpow(26,ans));
    return 0;
}
```

T3 lunatic

题意

- 给出 n 条线段和每条线段的左右端点 l_i, r_i
- 求将这些线段分为 K 组（不为空）的每一组的交集的最大和
- $1 \le n \le 10^5, 0 \le l_i < r_i \le 10^6, 1 \le k \le n$

分析

- 首先，如果一条线段 p_1 包含了另一条线段 p_2，那么在一组中，不论选不选 p_1，p_2 的贡献都不会变
  - 于是我们考虑先将包含了其他线段的线段单独分到类别 A 中，其他线段分到类别 B 中，这样在 A 中选 k' 条线段的最大贡献就是前 k' 长的线段的长度之和（选出 k' 条线段单独成组，其他线段与被他包含的线段成组）
  - 再在 B 中配出 k 组，于是总贡献就是这 k 组的贡献加上 A 中前 $K-k$ 长的线段长度之和
- 现在考虑如何求出这个最大贡献
  - 先将 A 中的线段按长度降序排序，将 B 中的线段按左端点升序排序（因为 B 中线段是互不包含的，所以右端点也是升序排序的）
  - 所以在 B 中所取的成组的线段一定是编号连续的线段（否则如果在中间取一些线段成组，两边再合在一起后的组的贡献就是负数（是负数的原因见后面），而有负数的分组方案是不优的，原因之后再讲）
  - 将 B 中每个组的第一条线段的编号设为 x_i（最后一组的最后一条线段设为 lst），B 的贡献就是

    $$\max(r_{x_1} - l_{x_2-1}, 0) + \max(r_{x_2} - l_{x_3-1}, 0) + \max(r_{x_3} - l_{x_4-1}, 0) + \cdots + \max(r_{x_k} - l_{lst}, 0)$$

  - 同时我们还要注意到另一种选取方式，即直接选出前 $K-1$ 长的线段单独成组，剩下的线段直接丢到一组中（将其贡献视作 0），于是我们就得到了答案的下界
  - 于是如果 B 中选的组有一组 $r_{x_i} - l_{x_2-1} < 0$，因为剩下的 $k-1$ 组的贡献加上 A 中 $K-k$ 条线段这一共 $K-1$ 组的贡献一定是小于等于下界的，所以这样选就没有下界优，他就不会作为最后的答案（上述第 2 点的原因也是如此）。所以我们就可以将 $\max$ 去掉

    $$r_{x_1} - l_{x_2-1} + r_{x_2} - l_{x_3-1} + r_{x_3} - l_{x_4-1} + \cdots + r_{x_k} - l_{lst}$$

  - 观察发现，我们将 $-l_{x_i-1} + r_{x_i}$ 作为一组，那在 B 中分 k 组的最大贡献就是选出前 $k-1$ 大的 $-l_{i-1} + r_i$，再加上 $r_1 - l_{lst}$，可以贪心解决
- 于是总过程就是求下界、分 A, B、排序、贪心

代码

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

const long long N=1e5+10;
long long n,m,A[N],cnta,B[N],cntb,C[N],ans;

struct line
{
```

```
    long long l,r;
}a[N];

bool cmp0(line x,line y)
{
    return x.r-x.l>y.r-y.l;
}

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

bool cmp2(long long x,long long y)
{
    return a[x].r-a[x].l>a[y].r-a[y].l;
}

int main()
{
    scanf("%lld%lld",&n,&m);
    for(long long i=1;i<=n;++i)
        scanf("%lld%lld",&a[i].l,&a[i].r);
    sort(a+1,a+n+1,cmp0);
    for(long long i=1;i<m;++i)                  //求下界
        ans+=a[i].r-a[i].l;
    sort(a+1,a+n+1,cmp);
    for(long long i=1;i<=n;++i)                 //用单调栈求A、B
    {
        while(cntb&&a[B[cntb]].r>a[i].r)
            A[++cnta]=B[cntb--];
        B[++cntb]=i;
    }
    for(long long i=2;i<=cntb;++i)              //C存储-l_{i-1}+r_{i}
        C[i]=a[B[i]].r-a[B[i-1]].l;
    sort(C+2,C+cntb+1,greater<long long>());
    sort(A+1,A+cnta+1,cmp2);
    long long tmp=a[B[1]].r-a[B[cntb]].l;
    for(long long i=1;i<m;++i)
        tmp+=a[A[i]].r-a[A[i]].l;
    ans=max(ans,tmp);
    for(long long i=1;i<m;++i)                  //贪心取最大值
    {
        tmp-=a[A[m-i]].r-a[A[m-i]].l;
        tmp+=C[i+1];
        ans=max(ans,tmp);
    }
    printf("%lld",ans);
    return 0;
}
```