第三周直播课（上午）作业讲解

洛谷网校

基础-提高衔接计划

2024-08

disangan233

www.luogu.com.cn

【第三周】异或

- T486175
- 有一个长度为 n 的正整数序列 a，找出一个子区间 $[l, r]$，使得 $a_l \operatorname{xor} a_{l+1} \operatorname{xor} \cdots \operatorname{xor} a_r$ 的结果最大
- $n \leq 10^3$，$a_i \leq 10^9$

【第三周】异或

- T486175
- 有一个长度为 n 的正整数序列 a，找出一个子区间 $[l, r]$，使得 $a_l \operatorname{xor} a_{l+1} \operatorname{xor} \cdots \operatorname{xor} a_r$ 的结果最大
- $n \le 10^3$，$a_i \le 10^9$
- 做法一：前缀和
  - 异或也具有可差分性，维护前缀和 b，区间和 $b_r \operatorname{xor} b_{l-1}$
- 做法二：双指针
  - 也可以不用前缀和，用双指针的思路，枚举 l，一个一个插入求异或和
- 时间复杂度都是 $O(n^2)$
- 虽然本题暴力常数很小，$O(n^3)$ 也能通过

【第三周】异或

```
scanf("%d",&n);
for(int i=1;i<=n;i++)
    scanf("%d",a+i);
for(int i=1;i<=n;i++)
    a[i]^=a[i-1];
for(int i=1;i<=n;i++)
    for(int j=i;j<=n;j++)
        ans=max(ans,a[j]^a[i-1]);
cout<<ans;
```

【第三周】我写前缀和！

- T486176
- $n \times n$ 的矩阵中有 m 个点，第 i 个点位于 (x_i, y_i)，求有多少个子矩阵至少包含 K 个点
- 对于所有数据，$n, m, K \leq 100$

【第三周】我写前缀和！

- T486176
- $n \times n$ 的矩阵中有 m 个点，第 i 个点位于 (x_i, y_i)，求有多少个子矩阵至少包含 K 个点
- 对于所有数据，$n, m, K \leq 100$
- 二维前缀和例题
- 枚举左上角 (i, j)，枚举右下角 (k, l)，使用前缀和求区间和
- 时间复杂度 $O(n^4)$

【第三周】我写前缀和！

```
scanf("%d%d%d",&n,&m,&K);
for(int i=1;i<=m;i++) {
    int x,y;
    scanf("%d%d",&x,&y);
    a[x][y]=1;
}
for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
        a[i][j]+=a[i-1][j]+a[i][j-1]-a[i-1][j-1];
for(int i=1;i<=n;i++) for(int j=1;j<=n;j++)
for(int k=i;k<=n;k++) for(int l=j;l<=n;l++)
    if(a[k][l]-a[i-1][l]-a[k][j-1]+a[i-1][j-1]>=K)
        ans++;
cout<<ans;
```

【第三周】我写前缀和吗？

- T486177
- $n \times n$ 的矩阵中有 m 个点，第 i 个点位于 (x_i, y_i)，求有多少个子矩阵至少包含 K 个点
- 对于所有数据，$n, m, K \leq 500$
- 如何优化时间复杂度？

【第三周】我写前缀和吗？

- 在区间和 $\geq K$ 之后继续枚举实质上是在浪费时间
- 所以可以只枚举 (i,j) 和 k，用双指针维护每个 j 所对应的最小的 l
- 由于此时 $(i,j) \to (k,l)$ 内已经至少包含 K 个点，对于当前的 j，$(k,l+1),(k,l+2),\dots(k,n)$ 都是满足条件的右下角
- 累加上对答案的贡献 $n-l+1$ 后，继续双指针枚举即可
- 时间复杂度 $O(n^3)$

【第三周】我写前缀和吗？

```
int ask(int i,int j,int k,int l) {
    return a[k][l]-a[i-1][l]-a[k][j-1]+a[i-1][j-1];
}
scanf("%d%d%d",&n,&m,&K);
for(int i=1;i<=m;i++) {
    int x,y;scanf("%d%d",&x,&y);
    a[x][y]=1;
}
for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
        a[i][j]+=a[i-1][j]+a[i][j-1]-a[i-1][j-1];
for(int i=1;i<=n;i++)
    for(int k=i;k<=n;k++)
        for(int j=1,l=1;j<=n;j++) {
            while(ask(i,j,k,l)<K&&l<=n) l++;
            if(l>n) break;
            ans+=n-l+1;
        }
cout<<ans;
```

【第三周】子序列

- T486179
- 给定两个长度为 n 和 m 的字符串 a, b，若字符串 a 的前缀为 b 的子序列，求这个前缀的最长长度
- 对于所有数据，$n, m \leq 2 \times 10^5$

【第三周】子序列

- T486179
- 给定两个长度为 n 和 m 的字符串 a, b，若字符串 a 的前缀为 b 的子序列，求这个前缀的最长长度
- 对于所有数据，$n, m \leq 2 \times 10^5$
- 双指针例题
- 维护双指针 i, j，用 a_i 去匹配 b_j，匹配到 $a_i = b_j$ 后 $i \leftarrow i+1$
- 若 a_i 失配（$j > m$），答案即为 $i-1$
- 例如：$a =$ abc，$b =$ acbda
- a_1 匹配 b_1，a_2 匹配 b_3，a_3 失配

【第三周】子序列

```
scanf("%d%d%s%s",&n,&m,a+1,b+1);
for(int i=1,j=1;i<=n;i++,j++) {
    while(a[i]!=b[j]&&j<=m) j++;
    if(j>m) {
        cout<<i-1;
        return 0;
    }
}
```

【第三周】区间

- T486180
- 给出 n 个区间 $[a_i, b_i]$，求出被区间覆盖次数最多的位置的被覆盖次数
- 对于所有数据，$n, a_i, b_i \leq 10^6$

【第三周】区间

- T486180
- 给出 n 个区间 $[a_i, b_i]$，求出被区间覆盖次数最多的位置的被覆盖次数
- 对于所有数据，$n, a_i, b_i \leq 10^6$
- 差分例题
- 对于区间 $[l, r]$ 加 1，令 $f_l \leftarrow f_l + 1$，$f_{r+1} \leftarrow f_{r+1} - 1$
- 做一次前缀和，统计序列最大值

【第三周】区间

```
scanf("%d",&n);
for(int i=1;i<=n;i++) {
    int x,y;
    scanf("%d%d",&x,&y);
    a[x]++;
    a[y+1]--;
}
for(int i=1;i<N;i++) {
    a[i]+=a[i-1];
    ans=max(ans,a[i]);
}
cout<<ans;
```

最大正方形

- P1387
- 求 $n \times m$ 的 01 矩阵中最大的不包含 0 的正方形，输出边长
- 对于所有数据，$n, m \leq 100$

最大正方形

- P1387
- 求 $n \times m$ 的 01 矩阵中最大的不包含 0 的正方形，输出边长
- 对于所有数据，$n, m \leq 100$
- 做法一：参考【第三周】我写前缀和！
- 维护二维前缀和，枚举左上角和右下角，求区间和判断是否等于区间面积大小
- 时间复杂度 $O(n^4)$
- 做法二：优化枚举
- 只枚举左上角和边长，时间复杂度 $O(n^3)$

最大正方形

```
scanf("%d%d",&n,&m);
for(int i=1;i<=n;i++)
for(int j=1;j<=m;j++) {
    scanf("%d",&a[i][j]);
    a[i][j]=a[i-1][j]+a[i][j-1]-a[i-1][j-1]+!(a[i][j]);
}
int ans=-1;
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        for(int p=1;p<=min(n-i+1,m-j+1);p++) {
            int k=i+p-1,l=j+p-1;
            if((a[k][l]-a[i-1][l]-a[k][j-1]+a[i-1][j-1])
                == 0)
                ans=max(ans,p);
        }
printf("%d\n",ans);
```

[TJOI2010] 阅读理解

- P3879
- 有 N 篇短文，每篇短文含 L 个单词
- 给定 M 个生词，求每个生词在哪些短文中出现过
- $N \le 10^3$，$M \le 10^4$，$L \le 5 \times 10^3$，$|s| \le 20$

[TJOI2010] 阅读理解

- P3879
- 有 N 篇短文，每篇短文含 L 个单词
- 给定 M 个生词，求每个生词在哪些短文中出现过
- $N \le 10^3$，$M \le 10^4$，$L \le 5 \times 10^3$，$|s| \le 20$
- 用 `set`、`map` 等数据结构存储每个短文的 L 个 Hash 值
- 对于一个生词，求出其 Hash 值，按顺序在数据结构中查找
- 要注意的是，`vector` 的 `find` 是 $O(n)$ 的

[TJOI2010] 阅读理解

```
set<pair<int,int>>S[1005];
scanf("%d",&n);
for(int i=1;i<=n;i++) {
    scanf("%d",&l);
    for(int j=1;j<=l;j++) {
        scanf("%s",s);
        S[i].insert(make_pair(h1(s),h2(s)));
    }
}
scanf("%d",&m);
while(m--) {
    scanf("%s",s);
    pair<int,int>h=make_pair(h1(s),h2(s));
    for(int i=1;i<=n;i++)
        if(S[i].count(h))
            printf("%d ",i);
    printf("\n");
}
```

[yLCPC2024] A. dx 分计算

- P10233
- T 组测试，每次给定一个字符串 s，q 组询问，每组询问给定 l, r，求 $s_{l...r}$ 的分数
- 定义一个字符串 s 的分数为 $3 \times \text{cnt}(\mathsf{P}) + 2 \times \text{cnt}(\mathsf{p}) + \text{cnt}(\mathsf{g})$，$cnt(x)$ 为字符 x 在 s 中的出现次数
- $\sum |s| \le 10^7$

[yLCPC2024] A. dx 分计算

- P10233
- T 组测试，每次给定一个字符串 s，q 组询问，每组询问给定 l, r，求 $s_{l \dots r}$ 的分数
- 定义一个字符串 s 的分数为 $3 \times \mathrm{cnt}(\mathsf{P}) + 2 \times \mathrm{cnt}(\mathsf{p}) + \mathrm{cnt}(\mathsf{g})$，$cnt(x)$ 为字符 x 在 s 中的出现次数
- $\sum |s| \le 10^7$
- 前缀和例题 2
- 为什么可以用前缀和来做？因为字符串的子串所含字符也是类似区间和的区间信息
- 根据读入的字符记录当前位的值，求前缀和
- 记得多测要清空（不要清空整个数组）

[yLCPC2024] A. dx 分计算

```
mp['P']=3,mp['p']=2,mp['g']=1;
while(T--) {
    scanf("%s",s+1);
    n=strlen(s+1);
    // memset(a+1,0,4*n);
    for(int i=1;i<=n;i++)
        a[i]=a[i-1]+mp[s[i]];
    scanf("%d",&q);
    while(q--) {
        int l,r;
        scanf("%d%d",&l,&r);
        printf("%d\n",a[r]-a[l-1]);
    }
}
```

[NOIP2012 提高组] 借教室

- P1083
- 给定 n 天的借教室信息，第 i 天有 r_i 个教室可供租借
- 有 m 份订单，第 i 份订单为某租借者需要在 s_i 天到 t_i 租借 d_i 个教室
- 按订单的先后顺序依次为每份订单分配教室，如果遇到一份订单无法完全满足，则停止分配，通知申请人修改订单
- 询问是否有订单无法被满足，如果有，给出通知的申请人编号
- $n, m \le 10^6$，$r_i, d_i \le 10^9$

[NOIP2012 提高组] 借教室

- 一份订单相当于在 $[s_i, t_i]$ 区间减去 d_i，可以用差分来维护
- 但是差分只能处理最后询问一次的情况，即是否有订单无法被满足
- 处理一次复杂度为 $O(n+m)$
- 考虑如果无法满足，如何找到是哪一个订单导致的
- 若第 i 天已经无法满足（有值为负），那么第 $i+1$ 天也无法满足
- 相当于查找负数第一次出现的订单号，二分答案即可
- 时间复杂度 $O((n+m)\log m)$

[NOIP2012 提高组] 借教室

```
bool check(int x) {
    for(int i=1;i<=n;i++) b[i]=a[i]-a[i-1];
    for(int i=1;i<=x;i++) b[l[i]]-=d[i],b[r[i]+1]+=d[i];
    for(int i=1;i<=n;i++) {
        b[i]+=b[i-1];
        if(b[i]<0) return 0;
    }
    return 1;
}
```

[NOIP2012 提高组] 借教室

```
int L=1,R=m;
if(check(m)) cout<<0;
else {
    while(L<R) {
        int mid=(L+R)>>1;
        if(check(mid)) L=mid+1;
        else R=mid;
    }
    cout<<"-1"<<endl<<L;
}
```

[USACO2.3] 最长前缀 Longest Prefix

- P1470
- 给定字符串集合 P，和字符串 s，求 s 的最长前缀 s'，使得 s' 可以被 P 中字符串拼接出，输出最长前缀长度
- $|P| \leq 200$，$|s| \leq 2 \times 10^5$，对于 P 中的字符串 t，$|t| \leq 10$

[USACO2.3] 最长前缀 Longest Prefix

- 考虑 dp，令 f_i 表示长度为 i 的前缀是否能被拼接出，有

$$f_i = \max_{j<i}\{f_j \text{ and } \operatorname{find}(s_{j+1\ldots i})\}(i \geq 1), \quad f_0 = 1$$

- $\operatorname{find}(s_{j+1\ldots i})$ 表示子串 $s_{j+1\ldots i}$ 是否属于集合 P，使用 Hash 来判断
- 直接 Hash 复杂度为 $O(|s| \times |t|^2)$，每一次求出子串的 Hash 值然后查找
- 使用对子串求 Hash 的技巧可以优化至 $O(|s| \times |t|)$
- 预处理前缀的 Hash 值和 B 的次幂，使用前缀求子串 Hash
- 注意题目给的是多行字符串

多次询问子串哈希

- 单次计算一个字符串的哈希值复杂度是 $O(n)$，与暴力匹配没有区别，如果需要多次询问一个字符串的子串的哈希值，每次重新计算效率非常低下
- 一般采取的方法是对整个字符串先预处理出每个前缀的哈希值，将哈希值看成一个 b 进制的数对 M 取模的结果，这样的话每次就能快速求出子串的哈希了：
- 令 $f_i(s)$ 表示 $f(s_{1..i})$，即原串长度为 i 的前缀的哈希值，那么按照定义有 $f_i(s) = s_1 \cdot b^{i-1} + s_2 \cdot b^{i-2} + \cdots + s_{i-1} \cdot b + s_i$
- 现在，我们想要用类似前缀和的方式快速求出 $f(s_{l..r})$，按照定义有字符串 $s_{l..r}$ 的哈希值为
$f(s_{l..r}) = s_l \cdot b^{r-l} + s_{l+1} \cdot b^{r-l-1} + \cdots + s_{r+1} \cdot b + s_r$

多次询问子串哈希

- 对比观察上述两个式子，发现 $f(s_{l..r}) = f_r(s) - f_{l-1}(s) \times b^{r-l+1}$ 成立（可以手动代入验证一下），因此我们用这个式子就可以快速得到子串的哈希值
- 其中 b^{r-l+1} 可以 $O(n)$ 预处理，然后 $O(1)$ 的回答每次询问（当然也可以快速幂 $O(\log n)$ 的回答每次询问）

[USACO2.3] 最长前缀 Longest Prefix

```
int n,m,ans,pw[N],h[N],f[N];
char s[N];
unordered_map<int,int>mp;
int main() {
    while(1) {
        scanf("%s",s+1);
        if(s[1]=='.') break;
        int cur=0,l=strlen(s+1);
        for(int i=1;i<=l;i++)
            cur=(1ll*cur*B+s[i])%M;
        mp[cur]=1;
    }
    char *t=s+1;
    while(scanf("%s",t)!=-1)
        t+=strlen(t);
```

[USACO2.3] 最长前缀 Longest Prefix

```
    n=strlen(s+1);
    pw[0]=1;
    for(int i=1;i<=n;i++) {
        h[i]=(1ll*h[i-1]*B+s[i])%M;
        pw[i]=1ll*pw[i-1]*B%M;
    }
    f[0]=1;
    for(int i=1;i<=n;i++)
    for(int j=max(0,i-10);j<i;j++)
    if(mp.count((h[i]-1ll*h[j]*pw[i-j]%M+M)%M))
        f[i]|=f[j];
    for(int i=1;i<=n;i++)
        if(f[i]) ans=i;
    cout<<ans;
}
```

魔族密码

- P1481
- 给定 n 个字符串 s_n，求其最长上升子序列长度
- 字符串序列 $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ 是上升子序列，当且仅当对于所有 $j < k$，满足 s_{i_j} 是 $s_{i_{j+1}}$ 的前缀
- $n \leq 2000$，$|s_i| \leq 75$

魔族密码

- P1481
- 给定 n 个字符串 s_n，求其最长上升子序列长度
- 字符串序列 $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ 是上升子序列，当且仅当对于所有 $j < k$，满足 s_{i_j} 是 $s_{i_{j+1}}$ 的前缀
- $n \le 2000$，$|s_i| \le 75$
- 做法同 LIS，令 f_i 为前 i 个字符串的最长上升子序列长度，有

$$f_i = \max_{s_j \text{ 是} s_i \text{ 的前缀}} \{f_j + 1\}$$

- 使用哈希来判断，时间复杂度 $O(n^2)$

魔族密码

```
scanf("%d",&n);
for(int i=1;i<=n;i++) {
    scanf("%s",s[i]+1);
    f[i]=1;
    l[i]=strlen(s[i]+1);
    for(int j=1;j<=l[i];j++)
        h[i][j]=(1ll*h[i][j-1]*B+s[i][j])%M;
    for(int j=i-1;j>=1;j--)
    if(h[i][l[j]]==h[j][l[j]])
        f[i]=max(f[j]+1,f[i]);
    ans=max(f[i],ans);
}
cout<<ans;
```

[Poetize6] IncDec Sequence

- P4552
- 给定长为 n 的数列 a_n，每次可以选择一个区间 $[l, r]$，使得区间内的数加 1 或减 1
- 求至少需要多少次操作才能使数列中所有数都一样，并求出次数最少的情况下，最终得到的数列有多少种
- $n \leq 10^5$，$0 \leq a_i \leq 2^{31}$

[Poetize6] IncDec Sequence

- 维护其差分数列 b，每一次相当于对一个位置 $+1$、一个位置 -1 或只对一个位置 ± 1（修改 $[i, n]$）
- 所有数都一样即除了 b_1 均为 0，需要把正数和负数都变为 0
- 为了让次数变少，应该优先让正数和负数相互抵消
- 正数需要的 -1 为 $b_2, \ldots, b_n$ 中所有正数之和次
- 负数需要的 $+1$ 为 $b_2, \ldots, b_n$ 所有负数之和的绝对值次
- 令其分别为 x, y，次数最少为 $\max(x, y)$
- 如果正数更多，在 b_1 可以加 $[0, |x - y|]$ 内的任意整数
- 如果负数更多，在 b_1 可以减 $[0, |x - y|]$ 内的任意整数
- 因此，数列种类为 $|x - y| + 1$ 种
- 样例的例子是 $1, 0, 1$，此时 $x = 1, y = 0$，答案为 $1, 2$

[Poetize6] IncDec Sequence

```
scanf("%d",&n);
for(int i=1;i<=n;i++)
    scanf("%lld",a+i);
for(int i=1;i<=n;i++)
    b[i]=a[i]-a[i-1];
for(int i=2;i<=n;i++) {
    if(b[i]>=0) x+=b[i];
    else y-=b[i];
}
cout<<max(x,y)<<"\n"<<abs(x-y)+1;
```

[NOI Online 2021 提高组] 积木小赛

- P7469
- 给定两个长度为 n 的字符串 s 和 t，求 t 中有多少本质不同的子串是 s 的子序列
- $n \leq 3000$

[NOI Online 2021 提高组] 积木小赛

- P7469
- 给定两个长度为 n 的字符串 s 和 t，求 t 中有多少本质不同的子串是 s 的子序列
- $n \le 3000$
- t 的子串数为 $O(n^2)$，子序列可以通过双指针来判断
- 具体的双指针实现：枚举 i, j，若 $t_{i...j}$ 是 s 的子序列，继续匹配 t_{j+1}，直到失配为止
- 类似【第三周】子序列的做法
- 用 Hash 来判重，时间复杂度 $O(n^2)$ 或 $O(n^2 \log n)$

[NOI Online 2021 提高组] 积木小赛

```
scanf("%d%s%s",&n,a+1,b+1);
for(int i=1;i<=n;i++) {
    long long cur=0;
    for(int j=i,p=1;j<=n;j++) {
        while(p<=n&&a[p]!=b[j]) p++;
        if(p>n) break;
        p++;
        cur=(1ll*cur*B+b[j]-'a'+1)%M;
        t[++t[0]]=cur;
    }
}
sort(t+1,t+t[0]+1);
printf("%d\n",unique(t+1,t+t[0]+1)-t-1);
```