本周推荐

陈铭煊 Max.D.

子集卷积

简介

一般我们有如下一类的状压dp方程，如$dp[i]=\sum dp[j]*w[k]$ (i,j,k满足$j\lor k=i,j \land k=0$□这里符号表示按位与和按位或。

如果暴力枚举位的子集，那么效率是3^n的，难以承受。

实际上这个已经很接近一个FWT卷积的形式了，只不过是还要$j\land k=0$罢了。

我们改变这个条件为□j中1的个数+k中1的个数=i中1的个数，那么当我们为dp增加一个“1的个数”的维度时，问题迎刃而解□ $$ dp[cnt_i][i]=\sum_{(j|k)==i}dp[cnt_j][j]*w[cnt_i-cnt_j][k] $$ 注意这里cnt_i表示1的个数，或者说子集中的物品数目。这里cnt_i和i的二进制1的个数如果不等，这个dp或者w值会置为0。此时只要我们从小到大枚举cnt来做FWT就可以得到答案了，实际操作过程中，所有的dp都是点值形式，因此得到新的$dp[cnt_i]$只需要做cnt_i次对位乘；最后，再将所有的dp逆FWT变换回原值。

虽然牺牲了一定空间，但是时间被优化到了n次FWT+n^2次对位乘法的复杂度□$O((2^n*n)*n+n^2*2^n)=O(n^2*2^n)$□

例题

模板题□https://ac.nowcoder.com/acm/contest/5157/D

很容易从题目的形式看出来实际上就是对四个数列求三重卷积，第一重是$i|j$的子集卷积，第二重$(i|j)+k$的FFT/NTT□第三重是$((i|j)+k)\otimes h$的FWT的异或卷积。代码如下：

```
#include <bits/stdc++.h>

#define N 262144

using namespace std;

const int mod = 998244353, inv2 = 499122177;

int n;
int rev[N], lim, hib;
int A[N], B[N], C[N], D[N], popc[N];
int f[20][N], g[20][N], h[20][N];

inline int Add(int u, int v) { return (u += v) >= mod ? u - mod : u; }
```

```
inline void Inc(int &u, int v) { if ((u += v) >= mod) u -= mod; }

inline int fpm(int x, int y) {
    int r = 1;
    while (y) {
        if (y & 1) r = 1LL * x * r % mod;
        x = 1LL * x * x % mod, y >>= 1;
    }
    return r;
}

inline int read() {
    int x = 0;
    char ch = getchar();
    while (!isdigit(ch)) ch = getchar();
    while (isdigit(ch)) x = x * 10 + ch - '0', ch = getchar();
    return x;
}

void getrev(int len) {
    lim = 1, hib = -1;
    while (lim < len) lim <<= 1, ++hib;
    for (int i = 0; i < lim; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << hib);
}

void fwtOr(int *a, bool type) {
    for (int mid = 1; mid < lim; mid <<= 1)
        for (int i = 0; i < lim; i += (mid << 1))
            for (int j = 0; j < mid; ++j)
                if (type) Inc(a[i + mid + j], a[i + j]);
                else Inc(a[i + mid + j], mod - a[i + j]);
}

void fwtXor(int *a, bool type) {
    static int x, y;
    for (int mid = 1; mid < n; mid <<= 1)
        for (int len = mid << 1, i = 0; i < n; i += len)
            for (int j = 0; j < mid; ++j) {
                x = a[i + j], y = a[i + mid + j];
                a[i + j] = Add(x, y), a[i + mid + j] = Add(x, mod - y);
                if (!type)
                    a[i + j] = 1LL * inv2 * a[i + j] % mod,
                        a[i + mid + j] = 1LL * inv2 * a[i + mid + j] %
mod;
            }
}

void NTT(int *a, bool type) {
    for (int i = 0; i < lim; ++i)
```

```
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    static int x, y;
    for (int mid = 1; mid < lim; mid <<= 1) {
        int len = mid << 1, wn = fpm(3, (mod - 1) / len);
        for (int i = 0; i < lim; i += len)
            for (int j = 0, w = 1; j < mid; ++j, w = 1LL * w * wn % mod) {
                x = a[i + j], y = 1LL * w * a[i + mid + j] % mod;
                a[i + j] = Add(x, y), a[i + mid + j] = Add(x, mod - y);
            }
    }
    if (!type) {
        reverse(a + 1, a + lim);
        int inv = fpm(lim, mod - 2);
        for (int i = 0; i < lim; ++i)
            a[i] = 1LL * inv * a[i] % mod;
    }
}

int main() {
    n = read(), ++n;
    getrev(n + n - 1);
    for (int i = 0; i < lim; ++i) popc[i] = popc[i >> 1] + (i & 1);
    for (int i = 0; i < n; ++i) A[i] = read(), f[popc[i]][i] = A[i];
    for (int i = 0; i < n; ++i) B[i] = read(), g[popc[i]][i] = B[i];
    for (int i = 0; i < n; ++i) C[i] = read();
    for (int i = 0; i < n; ++i) D[i] = read();

    for (int i = 0; i <= 17; ++i)
        fwtOr(f[i], true), fwtOr(g[i], true);
    for (int sa = 0; sa <= 17; ++sa)
        for (int sb = 0; sb + sa <= 17; ++sb)
            for (int i = 0; i < lim; ++i)
                h[sa + sb][i] = (h[sa + sb][i] + 1LL * f[sa][i] * g[sb][i])
% mod;
    for (int i = 0; i <= 17; ++i)
        fwtOr(h[i], false);
    for (int i = 0; i < lim; ++i)
        A[i] = h[popc[i]][i];

    NTT(A, true), NTT(C, true);
    for (int i = 0; i < lim; ++i)
        A[i] = 1LL * A[i] * C[i] % mod;
    NTT(A, false);

    fwtXor(A, true), fwtXor(D, true);
    for (int i = 0; i < lim; ++i)
        A[i] = 1LL * A[i] * D[i] % mod;
    fwtXor(A, false);

    int Q = read();
```

```
    while (Q--) printf("%d\n", A[read()]);
    return 0;
}
```

龙鹏宇 Hardict

统计数列中上升子序列个数

Since you want to know more, here is an explanation with more details […]

对应数列$\{a_{i}\}_{i=1}^{n}$

考虑一个dp转移:$f[n]=1+\sum_{1\leq i <n,a[i] < a[n]}f[i]$

可以利用数组数组解决,可是一般数列中会出现相同数,需要预先离散化

这里有一个技巧,假设数列长度为n,可以令$b[i]=(n+1)*a[i]+n-i排序后利用b[]离散化即可得到严格上升下对应的rank\\若令b[i]=(n+1)*a[i]+i则可得到不严格上升的rank$

例题

2015-2016 6th BSUIR Open Programming Contest. Semifinal A题

题意:

给定数列$\{a_{i}\}_{i=1}^{n},\{b_{i}\}_{i=1}^{n},a_{i} \leq 10^{9},b_{i} \leq 10^{6}$,求所有$\{a_{i}\}_{i=1}^{n}$上升子序列的下标对应的$\{b_{i}\}_{i=1}^{n}$的子序列gcd的和

题解:

考虑$满足新数列\{p\}=\{a_{i} : d |b_{i}\}$,求解$\{p\}$的上升子序列个数$cnt_{d}$再乘上容斥系数$coef_{d}$

即可得到$ans=\sum_{d=1}^{maxb}cnt_{d}coef_{d},maxb=\max\limits_{1 \leq i\leq n}\{b_{i}\}$

而对于容斥系数,考虑容斥过程

分析每一个d的实际贡献,其在$e|d$时都会被统计,对d单独进行容斥$coef_{d}=\sum_{e|d}e\mu(\frac{d}{e})$

这里可以预处理所有非μ进行预处理

还要预处理$\{i : d |b_{i}\}$进行优化

```
#include <algorithm>
#include <cstdio>
#include <cstring>
```

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

using LL = long long;
const int MOD = 1e9 + 7;
const int MAXN = 1e5 + 5;
// Euler sieve
int prime[MAXN * 10], cnt_prime;
bool noprime[MAXN * 10];
int mu[MAXN * 10];
vector<int> nozero_mu;
void Euler_sieve(int n);
// main
int N, M, K;
int a[MAXN], b[MAXN], p[MAXN], coef[MAXN * 10];
LL discret[MAXN];
vector<int> fac[MAXN * 10], buck[MAXN * 10];
int solve(int d);

int main() {
  //
  cin >> N;
  for (int i = 1; i <= N; i++) {
    cin >> a[i];
    discret[i] = 1LL * (N + 1) * a[i] + N - i;
  }
  sort(discret + 1, discret + 1 + N);
  // K = unique(discret + 1, discret + 1 + N) - discret - 1;
  K = N;
  for (int i = 1; i <= N; i++)
    a[i] = lower_bound(discret + 1, discret + 1 + K,
                       1LL * (N + 1) * a[i] + N - i) -
           discret;
  // for (int i = 1; i <= N; i++) cout << a[i] << endl;

  int maxb = 0;
  for (int i = 1; i <= N; i++) {
    cin >> b[i];
    buck[b[i]].push_back(i);
    maxb = max(maxb, b[i]);
  }
  Euler_sieve(maxb + 5);
  for (int i = 1; i <= maxb; i++) {
    for (int k : nozero_mu) {
      int j = i * k;
      if (j > maxb) break;
      coef[j] = (1LL * coef[j] + mu[k] * i + MOD) % MOD;
    }
  }
```

```
  for (int i = 1; i <= maxb; i++) {
    for (int j = i; j <= maxb; j += i)
      for (int id : buck[j]) fac[i].push_back(id);
    sort(fac[i].begin(), fac[i].end());
  }

  int ans = 0;
  for (int i = 1; i <= maxb; i++)
    if (coef[i]) {
      int tmp = 1LL * coef[i] * solve(i) % MOD;
      // cout << solve(i) << endl;
      ans = (ans + tmp) % MOD;
    }
  cout << ans << endl;
  return 0;
}

class FenwickTree {
 private:
  int n;
  int a[MAXN];
  inline int lowbit(int x) { return x & (-x); }

 public:
  void init(int n0);
  void add(int pos, int key);
  int query(int pos);
};
FenwickTree FT;
int solve(int d) {
  M = 0;
  for (int id : fac[d]) {
    p[++M] = a[id];
    discret[M] = p[M];
  }
  sort(discret + 1, discret + 1 + M);
  for (int i = 1; i <= M; i++)
    p[i] = lower_bound(discret + 1, discret + 1 + M, p[i]) - discret;
  int cnt = 0;
  FT.init(M);
  for (int i = 1; i <= M; i++) {
    int tmp = FT.query(p[i]);
    cnt = (cnt + tmp + 1) % MOD;
    FT.add(p[i], tmp + 1);
  }
  return cnt;
}

void FenwickTree::init(int n0) {
  n = n0;
```

```
  fill(a, a + n + 3, 0);
}

void FenwickTree::add(int pos, int key) {
  while (pos <= this->n) {
    this->a[pos] = (this->a[pos] + key) % MOD;
    pos += lowbit(pos);
  }
}

int FenwickTree::query(int pos) {
  int res = 0;
  while (pos) {
    res = (res + this->a[pos]) % MOD;
    pos -= lowbit(pos);
  }
  return res;
}

void Euler_sieve(int n) {
  mu[1] = 1;
  for (int i = 2; i <= n; i++) {
    if (!noprime[i]) {
      prime[++cnt_prime] = i;
      mu[i] = -1;
    }
    for (int j = 1; j <= cnt_prime && i * prime[j] <= n; j++) {
      noprime[i * prime[j]] = true;
      if (i % prime[j] == 0) break;
      mu[i * prime[j]] = -mu[i];
    }
  }
  for (int i = 1; i <= n; i++)
    if (mu[i]) nozero_mu.push_back(i);
}
```

肖思炀 MountVoom

其他

如何快速判断一段数字是一个$1\sim n$的排列：

给$1\sim n$随机一个hash值，如果这一段数字和异或和$ = 1\sim n$的异或和，那么认为这段数字是一个排列。

zzh的教诲

数组第二维不要开二的整次幂，会比较慢。

霍尔定理

霍尔定理 (题目在补了在补了qwq)