2026 Autumn Training · 做题记录

isaunoya · 2026-09-22

CF1187F. Expected Square Beauty [done]

题意 独立均匀取整数 $X_i \in [l_i, r_i]$，求极大连续相等段数的平方期望。

数据范围 $1 \le n \le 2 \times 10^5$，$1 \le l_i \le r_i \le 10^9$。

Solution

考虑计算 $E[B^k]$，其中 $k \ge 1$，原题对应 $k = 2$。下标从 0 开始，取值区间转为 $[L_i, R_i)$，末尾补充哨兵区间 $[0, 1)$。记此时共有 N 个变量、$m = N - 1$ 条边，$\mathrm{il}_i = (R_i - L_i)^{-1}$。令 $Y_i = [X_i \ne X_{i+1}]$，则段数 $B = \sum_{i=0}^{m-1} Y_i$。

展开 B^k 时，每项至多涉及 k 条不同的边，而总边数为 m，故只需统计 $j \le q = \min(k, m)$ 条边的贡献。每个选中连续段的长度不超过总选边数 j，因此连续段长度也只需计算到 q。

先求 $w_{l,t}$，表示边段 $[l, l+t)$ 全部满足 $Y_i = 1$ 的概率。

连续段概率的容斥

变量区间 $[s, r)$ 全部相等的概率为

$$c(s, r) = \max\Big(0, \min_{s \le i < r} R_i - \max_{s \le i < r} L_i\Big) \prod_{i=s}^{r-1} \mathrm{il}_i.$$

令 $r = l + t + 1$。在相等事件的容斥展开中，按包含 X_{r-1} 的连通块 $[s, r)$ 分类。块内 $r - s - 1$ 条边均选入容斥集合；若 $s > l$，分隔边 $s - 1$ 未选。因此

$$w_{l,t} = \sum_{s=l}^{r-1} (-1)^{r-s-1} c(s, r) \begin{cases} 1 & s = l \\ w_{l,s-l-1} & s > l \end{cases}.$$

初值 $w_{l,0} = 1$。未选入容斥集合的边不施加相等约束。

按 t 递增计算，固定 r 后向左枚举 s，维护区间交集与逆元乘积，即可常数时间更新 $c(s, r)$。

再求 $\mathrm{dp}_{i,j}$，表示边前缀 $[0, i)$ 中所有大小为 j 的边集同时变化的概率之和。将选中的边分解为极大连续段，不同段之间至少空出一条边，依赖的原变量集合不交，因此联合概率为各段概率之积。

DP 转移

按末边是否被选分类。若被选，枚举最后一个极大连续段的长度 t，其前一条边必须未选：

$$\mathrm{dp}_{i,j} = \mathrm{dp}_{i-1,j} + \sum_{t=1}^{\min(i,j)} w_{i-t,t} \begin{cases} [j = t] & i = t \\ \mathrm{dp}_{i-t-1,j-t} & i > t \end{cases}.$$

初值 $\mathrm{dp}_{0,0} = 1$，其余状态为 0。

令 $a_j = \mathrm{dp}_{m,j} = E\Big[\binom{B}{j}\Big]$。指示变量的重复幂可以合并，按不同下标的数量分组，用第二类 Stirling 数还原：

$$E[B^k] = \sum_{j=0}^{q} S(k, j) j! a_j.$$

其中 $S(k,j)=S(k-1,j-1)+jS(k-1,j)$，$S(0,0)=1$，其余边界状态为 0。代码用 s_j 倒序滚动计算，最后得到 $s_j=S(k,j)$。

总时间为 $O(mq^2+kq)$，空间为 $O(mq)$。原题 $k=2$ 时为线性规模。

```cpp
#include "noya/head.hpp"

#include "noya/modint.hpp"

using namespace noya;

using mi = mint107;

vc<vc<mi>> block_probabilities(const vl &L, const vl &R, int k) {
  int N = sz(L), m = N - 1, q = min(k, m);
  vc<mi> il(N);
  rep(i, N) il[i] = mi(R[i] - L[i]).inv();

  vc<vc<mi>> w(m, vc<mi>(q + 1));
  rep(l, m) {
    w[l][0] = 1;
    rep(t, 1, min(q, m - l) + 1) {
      int r = l + t + 1;
      ll ml = L[r - 1], mr = R[r - 1];
      mi prod = 1;
      for (int s = r - 1; s >= l; s--) {
        cmax(ml, L[s]);
        cmin(mr, R[s]);
        prod *= il[s];
        mi c = mi(max(mr - ml, ll(0))) * prod;
        mi pre = s == l ? mi(1) : w[l][s - l - 1];
        if ((r - s - 1) & 1)
          w[l][t] -= pre * c;
        else
          w[l][t] += pre * c;
      }
    }
  }
  return w;
}

vc<mi> factorial_moments(const vc<vc<mi>> &w, int k) {
  int m = sz(w), q = min(k, m);
  vc<vc<mi>> dp(m + 1, vc<mi>(q + 1));
  dp[0][0] = 1;
  rep(i, 1, m + 1) {
    dp[i] = dp[i - 1];
    rep(j, 1, min(i, q) + 1) {
      rep(t, 1, min(i, j) + 1) {
        int l = i - t;
        mi pre = l == 0 ? mi(j == t) : dp[l - 1][j - t];
        dp[i][j] += pre * w[l][t];
      }
    }
  }
  return dp[m];
}
```

```
mi expected_moment(const vl &L, const vl &R, int k) {
  assert(k >= 0 && !L.empty() && sz(L) == sz(R));
  auto w = block_probabilities(L, R, k);
  auto a = factorial_moments(w, k);
  int q = sz(a) - 1;
  vc<mi> s(q + 1);
  s[0] = 1;
  rep(n, k) {
    for (int j = min(n + 1, q); j > 0; j--) {
      s[j] = s[j - 1] + mi(j) * s[j];
    }
    s[0] = 0;
  }
  mi fac = 1, ans = s[0] * a[0];
  rep(j, 1, q + 1) {
    fac *= j;
    ans += s[j] * fac * a[j];
  }
  return ans;
}

void solve(int k = 2) {
  int n;
  cin >> n;
  vl L(n), R(n);
  rep(i, n) cin >> L[i];
  rep(i, n) {
    cin >> R[i];
    R[i]++;
  }
  L.push_back(0);
  R.push_back(1);

  cout << expected_moment(L, R, k).val() << "\n";
}

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);
  solve();
  return 0;
}
```

ABC476G. Increasing Popcount [done]

题意 将 $\mathrm{popcount}(L), \ldots, \mathrm{popcount}(R)$ 划分成尽量少的严格递增子序列，求最少个数。

数据范围 $1 \le T \le 10^4$，$1 \le L \le R \le 10^{18}$。

Solution

由 Dilworth 定理，答案等于 popcount 序列的最长不升子序列长度，相等值可以连续选取。

令 $B = 60$，将输入区间转为 $[L, R)$。按照线段树查询的方式，从左到右拆成 $O(B)$ 个二进制对齐块。每次取不越过 R、且 L 为其长度倍数的最大块 $[L, L + 2^k)$。令 $c = \mathrm{popcount}(L)$，块内值为 $c + \mathrm{popcount}(x)$，其中 $0 \le x < 2^k$；值 $c + t$ 出现 $C_{k,t} = \binom{k}{t}$ 次。

块内只需考虑选取某个固定的 popcount 值。对于任意非空不升子序列，都能在其最小值与最大值之间找到一层，改选该层的全部元素，长度不减，且仍能与两侧拼接。

为什么块内只需选择一个值

忽略共同的高位贡献 c，对 k 归纳。$k=0$ 时结论显然成立。对于 $k>0$，将块均分为左右两半，右半的 popcount 比左半对应位置多 1。对非空的半块子序列应用归纳假设，可替换为该半块中某一值的全部元素，长度不减，且替换值仍在原子序列的值域内。

若只选了一半，直接在整块中选取同一个值，数量不会减少。否则，设左半选值 a，右半选值 b，必有 $a \geq b$，两半的数量分别为 $C_{k-1,a}$ 和 $C_{k-1,b-1}$。

令 $h = \left\lfloor \frac{k-1}{2} \right\rfloor$。由二项式系数的单峰性，当 $a > b$ 时，可以进行以下调整：

- 若 $a > h$，则 $C_{k-1,a-1} \geq C_{k-1,a}$，将左半所选值由 a 改为 $a-1$。
- 若 $a \leq h$，则 $b < a \leq h$，从而 $C_{k-1,b} \geq C_{k-1,b-1}$，将右半所选值由 b 改为 $b+1$。

每次调整后仍选取相应值的全部元素，所选数量不减，且仍有 $a \geq b$。同时，非负整数 $a-b$ 减少 1，因此有限次调整后必有 $a=b$。

最终二者相等，记为 v，此时选中的正是整个块中值为 v 的一层，其大小为

$$C_{k-1,v} + C_{k-1,v-1} = C_{k,v}.$$

所有调整后的值均落在初始的 $[b,a]$ 内，因而仍在原子序列的值域内，不影响与块外两侧拼接，归纳完成。

令 dp_j 为已处理前缀中末项至少为 j 的最长不升子序列长度，初值均为 0，表示空序列。追加一块时，选取值为 j 的整层，贡献为 $C_{k,j-c}$；再对末值取后缀最大值。按 $j = c+k, \ldots, 0$ 逆序更新：

$$\mathrm{dp}_j \leftarrow \max\left(\mathrm{dp}_j + C_{k,j-c}, \mathrm{dp}_{j+1}\right).$$

其中 $t<0$ 或 $t>k$ 时 $C_{k,t}=0$。更新时 dp_j 仍是旧状态，dp_{j+1} 已是新状态，可直接原地转移。最终答案为 dp_0。

预处理二项式系数需 $O(B^2)$ 时间与空间；每组需 $O(B^2)$ 时间、$O(B)$ 额外空间。

```cpp
#include "noya/head.hpp"

#include <bit>

using namespace noya;

constexpr int B = 60;
ll C[B + 1][B + 1];

void solve() {
  ll L, R;
  cin >> L >> R;
  R++;
  ll dp[B + 2]{};
  while (L < R) {
    int k = min(countr_zero(ull(L)), int(bit_width(ull(R - L))) - 1);
    int c = popcount(ull(L));
    for (int j = c + k; j >= 0; j--) {
      if (j >= c) dp[j] += C[k][j - c];
      cmax(dp[j], dp[j + 1]);
    }
    L += 1LL << k;
```

```
  }
  cout << dp[0] << "\n";
}

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);
  rep(k, B + 1) {
    C[k][0] = 1;
    rep(j, 1, k + 1) C[k][j] = C[k - 1][j - 1] + C[k - 1][j];
  }
  int t;
  cin >> t;
  while (t--) solve();
  return 0;
}
```