Problema 2. Matrice Interesantă

Autor: Lorintz Alexandru, Universitatea Babeș-Bolyai

Cuvinte cheie: Two pointers, Range minimum query, Măști de biți

Indicații

La o primă analiză a restricțiilor problemei, se observă că numerele din matrice au valori mici, ceea ce indică către o soluție care ar putea profita de această limitare.

Pentru început, se pot fixa liniile care limitează subamtricele căutate superior, respectiv inferior, obținându-se astfel **O(N^2)** perechi. Acum trebuie găsită o abordare eficientă pentru a număra perechi de coloane care împreună cu aceste linii determina submatrice cu proprietatea din enunț. Problema aici este că proprietatea din enunț este mult prea restrictivă, dar poate fi reformulată, calculând submatrice cu cel mult **X** elemente distincte, rezultatul căutat devenind astfel diferența între rezultatele noii probleme pentru **X=K** și **X=K-1**.

Pentru noua problemă definită, se poate folosi **tehnica celor doi pointeri** în continuare, dar este nevoie de o modalitate rapidă de a afla la fiecare pas câte elemente distincte se află într-o submatrice fixată, interogare necesară pentru modalitatea de actualizare a pointerilor. Aici intervine limitarea valorilor din matrice. Practic, pentru o submatrice se poate folosi o **mască de biți** care are un bit activ dacă numărul valoarea acelui bit apare în submatrice. Pentru a calcula această mască rapid pentru o submatrice arbitrară, se poate precalcula o structură de date similară cu un **Range minimum query bidimensional**. De asemenea, este de menționat că ne preocupă numărul de biți setați ai unei măști arbitrare, dar acesta poate fi ușor aflat folosind funcția predefinită **__builtin_popcountll**, care este foarte rapidă în practică. Astfel, complexitatea totală este **O(N^2 * log^2(N) + N^2 * M)** și **O(N^2 * log^2(N))** spațiu de la calcularea structurii de **Range minimum query**.

Soluție

```
#include <fstream>
#include <cstdint>
using namespace std;

ifstream fin("matriceinteresanta.in");
ofstream fout("matriceinteresanta.out");

const int kN = 300;
const int kLog = 8;
int n, m, a[1 + kN][1 + kN];
int lg2[1 + kN];
int64_t rmq[1 + kLog][1 + kLog][1 + kN][1 + kN];

int64_t query(int x1, int y1, int x2, int y2) {
    if (x2 < x1 || y2 < y1) {
        return 0;
    }

  int h1 = lg2[x2 - x1], h2 = lg2[y2 - y1];
  return rmq[h1][h2][x1][y1] | rmq[h1][h2][x2 - (1 << h1) + 1][y1] |
           rmq[h1][h2][x1][y2 - (1 << h2) + 1] | rmq[h1][h2][x2 - (1 << h1) + 1][y2 - (1
<< h2) + 1];
}

void precalc() {
```

```
  for (int i = 2; i <= kN; ++i) {
    lg2[i] = lg2[i >> 1] + 1;
  }

  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      rmq[0][0][i][j] = 1LL << a[i][j];
    }
  }

  for (int h1 = 0; h1 <= kLog; ++h1) {
    for (int h2 = 0; h2 <= kLog; ++h2) {
      if (h1 > 0 || h2 > 0) {
        for (int i = 1; i <= n - (1 << h1) + 1; ++i) {
          for (int j = 1; j <= m - (1 << h2) + 1; ++j) {
            rmq[h1][h2][i][j] = query(i, j, i + (1 << h1) - 1, j + (1 << h2) - 1);
          }
        }
      }
    }
  }
}

int64_t solveAtMostK(int k) {
  int64_t res = 0;

  for (int l1 = 1; l1 <= n; ++l1) {
    for (int l2 = l1; l2 <= n; ++l2) {
        int left = 1;

        for (int right = 1; right <= m; ++right) {
            while (left <= right && __builtin_popcountll(query(l1, left, l2, right)) > k)
{
                left += 1;
            }

            res += right - left + 1;
        }
    }
  }

  return res;
}

int main() {
  int k;
  fin >> n >> m >> k;

  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
        fin >> a[i][j];
    }
  }

  precalc();

  fout << solveAtMostK(k) - solveAtMostK(k - 1) << '\n';

  return 0;}
```