2023暑期CSP-S/NOIP模拟赛7 题解

T1 矩阵染色

首先的一个性质：4个区间面积的情况相等，直接计算一个区域最后将答案 × 4 就可以了。

40%的做法：

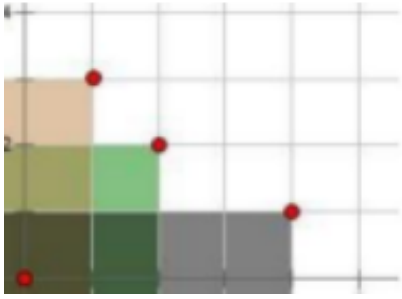
考虑坐标的大小，整个能覆盖到的范围不超过 3333 * 3333，也就是答案大小不超过这么多。

我们可以考虑判断 3333 * 3333 中的每个点是否包含某在一个矩阵中。

判断的方法：对 (x_i, y_i) 点对按照 x_i 进行排序，可以使用一个 vector 来记录能够覆盖到 x_i 的 y_i 有哪些。对每个点的查询都可以先二分 x_i ，找到能覆盖这个点最小的 x ，并查询能覆盖这个 x 最大的 y 是多少（这里可以发现记录 y 的 vector 可以舍弃，直接记录最大值的 y 即可）。

额外10%的做法：

没有包含的情况，那么所有的矩阵一定满足形式：以一个递降的形式。可以直接计算每一个矩阵相对于之前多出来的部分。



100%的做法：

发现有包含的情况也可以用类似的方法来做，只不过需要排除掉包含的情况。

对于 n <= 1e6 ，排除的方法可以使用二维数点的方式：对每个点查询左上角是否有其他的点，可以使用树状数组来实现。

```cpp
#include <bits/stdc++.h>
#define mk make_pair
#define x first
#define y second
using namespace std;
const int maxn = 1e6 + 10;
int n;
pair<int, int> a[maxn];
int Rank[maxn];
long long ans;
struct TA {
  int tr[maxn];
```

```
  TA() {
    memset(tr, 0, sizeof tr);
  }

  void update(int x) {
    while(x <= n) ++tr[x], x += x & -x;
  }

  int query(int x) {
    int tot = 0;
    while(x) tot += tr[x], x -= x & -x;
    return tot;
  }
}T;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i].x >> a[i].y;
    a[i].x /= 2, a[i].y /= 2;
    Rank[i] = a[i].y;
    }
  sort(a + 1, a + 1 + n);
  sort(Rank + 1, Rank + 1 + n);
  int Maxy = 0;
  for(int i = n; i > 0; --i) {
    int id = lower_bound(Rank + 1, Rank + 1 + n, a[i].y) - Rank;
    T.update(id);
    if(n - i - T.query(id) + 1 > 0) continue; //二维数点判断
    ans += 1ll * (a[i].y - Maxy) * a[i].x;
    Maxy = a[i].y;
  }
    cout << ans * 4 << '\n';
    return 0;
}
```

当然，观察最后形成的范围，都会是一段一段的，且是一个递降的形式。我们可以按照坐标从大往小考虑，直接维护每个坐标的高度，也可以获得答案。

```
#include <bits/stdc++.h>
using namespace std;
int n, Mx[10000010];
long long ans;
int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
```

```
    cin >> n;
    for (int x, y, i = 1; i <= n; i++) {
        cin >> x >> y;
        x /= 2, y /= 2;
        Mx[x] = max(Mx[x], y);
    }
    for (int h = 0, i = 10000000; i >= 1; i--) {
        h = max(h, Mx[i]);
        ans += h;
    }
    cout << ans * 4 << '\n';
    return 0;
}
```

T2 奇偶数

一个性质：模2的加法就是异或

35%的做法：

枚举边权为 0/1，使用 bfs 直接判断每个条件是否合理。

时间复杂度：$O(n*q*2^{n-1})$

额外15%~25%的做法：

对于一条链的情况且 q 很小的情况，应该如何考虑?

对于 q=1 的情况，他们只与从 u 到 v 的路径有关，而要让这一段的异或和为 0 或者 1 的方案数，就 $2^c/2$ （其中 c 为路径的边数），这是因为只有 0/1 两种答案，且两种答案是对半分配的。或者指定一条边用来专门调整数值为 0 还是 1，其他边随意。

对于 q=2 的情况，要稍微复杂一点，需要分情况讨论，他们是否有交集，按照 q=1 相同的方法来考虑。考虑相交部分和不相交部分分别为 0/1 的方案数，最后乘起来。

对于链的情况，处理起来会更简单一点。

100%的做法：

对于链的情况，与序列相似，是很容易到使用从根到端点异或前缀和来处理，将原本的条件 (u,v,x) 转为了 dis[u]^dis[v]=x，其中dis[i]表示i到根的边权的异或前缀和。

而在树上的边权路径异或和等于两个端点直接到根的异或和异或起来，因为相同的部分异或，会消掉。这也是满足 dis[u]^dis[v]=x 的条件。

这样就可以将问题转化为有 Q 个条件 dis[u]^dis[v]=x 需要满足，同时可以发现，这些等式与树的形状（也即具体路径上有哪些边）无关，只与起始点相关。可以令dis[1]=0，其他的边权都可以用dis值的异或表示出来。

这就是一个很经典的带权并查集或者扩展域并查集。

以带权并查集为例，如果两个点之间有条件限制，将两个点所在的并查集合并，使用对应的边权来表示边权，一个并查集中的点取值会相互限制。

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;
int n, q, ans = 1, sum, mod = 1e9 + 7;
int fa[1000001], d[1000001];

int find(int x) {
    if (fa[x] == x) return x;
    int t = fa[x];
    fa[x] = find(fa[x]);
    d[x] = d[x] ^ d[t];
    return fa[x];
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);
    cin >> n >> q;
    for (int i = 1; i <= n; ++i) fa[i] = i;
    for (int i = 1; i < n; ++i) {
        int u, v;
        cin >> u >> v;
    }
    for (int i = 1; i <= q; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        int x = find(u), y = find(v);
        if (x == y && d[u] != (d[v]^w)) {
            cout << "0";
            return 0;
        }
    else if (x != y) {
            fa[y] = x;
            d[y] = d[u] ^ d[v] ^ w; //合并带权并查集
        }
    }
    for (int i = 1; i <= n; ++i) if (find(i) == i) sum++;
    for (int i = 1; i < sum; ++i) ans = (ans * 2) % mod;
    cout << ans << '\n';
    return 0;
}
```

T3 任务

30%的做法：

依次枚举没给任务在哪个机器上处理。

时间复杂度 $O(2^n)$

70%的做法：

考虑在处理任务的时候，需要知道每台机器上的任务还需要多少时间完成。可以设状态为$dp[i][j][k]$表示处理完前i个任务，第一台机器上的任务还要j时间完成，第二台机器上的任务还需要k时间完成时的最小花费时间。

转移很简单：如果第i个任务，在$dp[i-1][j][k]$基础上完成，在第一台机器上完成的情况：
$dp[i][t1][t2] = min(dp[i][t1][t2], dp[i-1][j][k] + cost)$

- A机器需要的时间多，继续往A机器放任务，需要考虑特殊情况（见100%的做法）： If(j>=k) cost=A[i], t1=A[i], t2=0;
- B机器的时间多，此时的花费的最大时间由机器B决定：else if(j+A[i]<=k) cost=0, t1=j+A[i], t2=k;
- 普通情况：else cost=j+A[i]-k, t1=j+A[i]-k, t2=0;

在第二台机器上完成任务时，转移类似。

这个问题的复杂度是$O(n*m*m)$,m为完成某个任务的最长时间。

100%的做法：

在70%做法的基础上，考虑如何优化问题：在转移的时候，主要考虑两台机器的相对工作时间。可以用他们的差值来表示当前的状态（减少状态数）。

$dp[i][j]$ 表示完成前i个任务，第一台机器完成当前任务的时间比第二台多j的**第一台机器最小花费的时间**。第二台机器花费的时间可以通过 $dp[i][j] + aba(j)$ 来计算。

具体的转移方程见代码。

j可能为负数，加上单个任务的最大值M即可。

特殊情况：

如果第一台机器的完成时间已经比第二台要晚了，这时仍然把任务交个第一台机器的时候，两台机器的时间差就变为了A[i]，而不是在原有的j的基础上加上A[i],原因是需要等第一台的上一个任务完成，才能在第二台机器上开始任务，这样时间才不会错。如下图：

第一台	第二台
-	.
6	-
5	-
3	4
2	1
表格中的数字表示任务编号，如果第6个任务要在第一台机器上完成，那么在第二台机器上做任务的开始时间至少要等任务5完成。这样才能满足题目中的条件。	

```cpp
#include <bits/stdc++.h>
#define N 3005
using namespace std;
int n, ans = 0x7fffffff, t1[N], t2[N], f[N][N * 2 + 1];

signed main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);
    cin >> n;
  for(int i = 1; i <= n; ++i) cin >> t1[i] >> t2[i];
    memset(f, 63, sizeof(f));
    f[0][N] = 0;
  for(int i = 1; i <= n; ++i)
    for(int j = 0; j <= 2* N; ++j) {
      if (j >= N) {
                //对于 j >= 0，此时A机器花的时间多，继续往A机器上放任务的话，需要考虑特殊情况
        f[i][N + t1[i]] = min(f[i][N + t1[i]], f[i - 1][j] + t1[i]);
                //放在B机器上，由于只考虑A花费的时间，不加cost
        f[i][j - t2[i]] = min(f[i][j - t2[i]], f[i - 1][j]);
      }
      else {
                //B机器花的时间多，继续放在B机器上面，需要考虑特殊情况
        f[i][N - t2[i]] = min(f[i][N - t2[i]], f[i - 1][j] + N - j);
                //放在A机器上面正常处理
        f[i][j + t1[i]] = min(f[i][j + t1[i]], f[i - 1][j] + t1[i]);
      }
    }
  for(int i = 0; i <= 2 * N; ++i) ans = min(ans, f[n][i] + max(0, N - i));
  cout << ans << '\n';
  return 0;
}
```

T4 天分测试

性质：本质不同的起点和方向的选择只有 n * m * 8 个。

20%的做法：

同时枚举2个字符串的起点和方向，直接枚举K的长度，暴力判读是否一样。

时间复杂度：$O((8nm)^2 * k)$

40%的做法：

对每种起点和方向，记录对应的哈希值，记录有多少个哈希值与当前枚举起点和方向对应的字符串哈希值一样。

使用map来实现哈希的话，时间复杂度：$O(8nmk * log_2(nmk))$

70%的做法：

对于 n=m 的情况，由于所有方向的循环节都是n，这个时候只需要验证k=n的情况下相等，就可以得到所有k>n的情况了。

100%的做法：

直接计算哈希值，由于k的范围很大，只有 n*m 的范围是独特的，其他是由复制得到的，考虑使用倍增来加速求哈希的过程。

但是需要注意哈希的模数的选取，在模数取 998244353 或 10^9+7 时都会被卡掉一个点。

可以选择使用：

1. 使用双哈希。
2. 使用一些不那么有名的模数，例如：998244853

时间复杂度：$O(8nm\log_2 k)$

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;
char grid[220][220];
const int dx[] = {0, 0, 1, 1, 1, -1, -1, -1};
const int dy[] = {1, -1, 1, 0, -1, 1, 0, -1};
const ll mod = 998244853;
const ll base = 47;
ll Hash[220][220][8][32];
ll Power[33];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    Power[0] = base;
    for (int i = 1; i < 33; i++) Power[i] = Power[i - 1] * Power[i - 1] % mod;
    int n, m;
    ll k;
```

```
  cin >> n >> m >> k;
    for (int i = 0; i < n; i++) cin >> grid[i];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            for (int d = 0; d < 8; d++) {
                Hash[i][j][d][0] = (grid[i][j] - 'a' + 1);
            }
    //预处理倍增哈希
    for (int l = 1; l < 30; l++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                for (int d = 0; d < 8; d++) {
                    int nx = ((i + dx[d] * (1 << (l - 1))) % n + n) % n;
                    int ny = ((j + dy[d] * (1 << (l - 1))) % m + m) % m;
                    Hash[i][j][d][l] = (Hash[i][j][d][l - 1] * Power[l - 1] +
Hash[nx][ny][d][l - 1]) % mod;
                }
    vector<ll> vec;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            for (int d = 0; d < 8; d++) {
                ll Hsh = 0;
                int x = i, y = j;
                for (int l = 29; l >= 0; l--)
                    if (k >> l & 1) {
                        int nx = ((x + dx[d] * (1 << (l))) % n + n) % n;
                        int ny = ((y + dy[d] * (1 << (l))) % m + m) % m;
                        Hsh = (Hsh * Power[l] + Hash[x][y][d][l]) % mod;
                        x = nx;
                        y = ny;
                    }
                vec.push_back(Hsh);
            }
    sort(vec.begin(), vec.end());
    int cnt = 0;
    ll sum = 0;
    for (int i = 0; i < (int)(vec.size()); i++) {
        if (i > 0 && vec[i] == vec[i - 1]) cnt++;
        else {
            sum += 1ll * cnt * cnt;
            cnt = 1;
        }
    }
    sum += 1ll * cnt * cnt;
    ll deno = 1ll * ((int)(vec.size())) * ((int)(vec.size()));
    ll g = __gcd(deno, sum);
    deno /= g;
    sum /= g;
    printf("%lld/%lld\n", sum, deno);
```

```
    return 0;
}
```

补充作法：

对于 n!=m 的情况，循环节为 lcm(n,m)，k的范围只需要考虑到 lcm(n,m) 的大小即可。

对于哈希的做法，可以使用 unsigned long long 进行自然溢出。如果你得了80或者90分，那么通常是哈希的底数或者模数找的不够大，可以尝试使用更好的模数或者双哈希。

附上另外100分做法：

```
#include <bits/stdc++.h>
#define LL long long
#define Z(a,b,c) a*n*m+b*m+c+1
//使用 z 函数将 (n,m,d) 变为一维
using namespace std;
int n, m, k, g[25][320005], lg, x;
int dx[8] = {1, 1, 1, 0, 0, -1, -1, -1}, dy[8] = {-1, 0, 1, 1, -1, 1, 0, -1};
unsigned LL f[25][320005], ha[20], res;
LL A, t;
char s[210][210];
map<unsigned LL, int>mp;

LL gcd(LL x, LL y) {
    return !y ? x : gcd(y, x % y);
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);
    cin >> n >> m >> k;
  k = min(k, n * m / (int)gcd(n, m));
    lg = log2(t = n * m * 8);
    for (int i = 0; i < n; i++) cin >> s[i];
    for (int d = 0; d < 8; d++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++) {
                f[0][Z(d, i, j)] = s[i][j] - 'a';
        g[0][Z(d, i, j)] = Z(d, (i + dx[d] + n) % n, (j + dy[d] + m) % m);
      }
    ha[1] = 1e9 + 9;
    for (int i = 2; i <= lg; i++) ha[i] = ha[i - 1] * ha[i - 1];
    for (int i = 1; i <= lg; i++)
        for (int j = 1; j <= t; j++)
            f[i][j] = f[i - 1][j] * ha[i] + f[i - 1][g[i - 1][j]], g[i][j] = g[i - 1][g[i - 1][j]];
    for (int i = 1; i <= t; i++) {
        res = 0, x = i;
        for (int j = 0; j <= lg; j++)
```

```
            if (k & (1 << j))
                res = res * ha[j + 1] + f[j][x], x = g[j][x];
        A += 2ll * mp[res] + 1, mp[res]++;
    }
    LL tt = gcd(A, t * t);
    printf("%lld/%lld", A / tt, t * t / tt);
    return 0;
}
```