第 2 章

累積和

2.1 問題 B06：Lottery (難易度：★2相当)

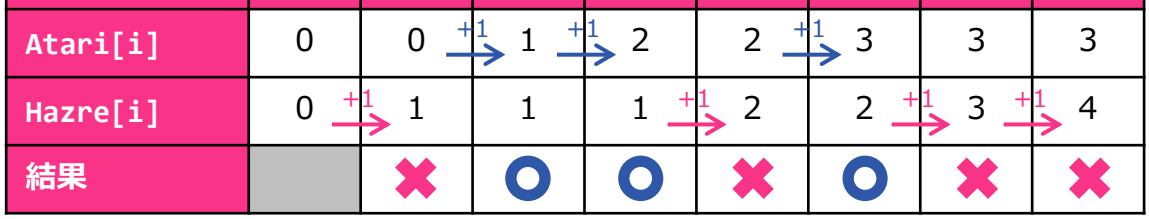
この問題を解く最も単純な方法は全探索です。L 回目から R 回目までのアタリの数・ハズレの数を for 文で数えると、計算量 $O(R-L)$ で「アタリとハズレどちらが大きいか」という質問に答えることができます。

しかし、本問題の制約は $N, Q \leq 100000$ であるため、残念ながら実行時間制限に間に合いません。一体どうすれば良いのでしょうか。

累積和を使おう

まず、**アタリの数・ハズレの数それぞれについて累積和をとる**ことを考えます。具体的には、1 回目から i 回目までのアタリの数 `Atari[i]`、1 回目から i 回目までのハズレの数 `Hazre[i]` を前もって計算します。入力例 1 のときの配列の値を以下に示します。

i	0	1	2	3	4	5	6	7
Atari[i]	0	0	1	2	2	3	3	3
Hazre[i]	0	1	1	1	2	2	3	4
結果		✖	○	○	✖	○	✖	✖



すると、L 回目から R 回目までのアタリの数・ハズレの数は、それぞれ以下のようにして計算することができます。

- アタリの数：`Atari[R]-Atari[L-1]`
- ハズレの数：`Hazre[R]-Hazre[L-1]`

したがって、次ページの解答例のような実装をすると、高速に答えを出すことができます。計算量は $O(N+Q)$ です。

解答例（C++）

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

int N, A[100009];
int Q, L[100009], R[100009];
int Atari[100009], Hazre[100009];

int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> A[i];
    cin >> Q;
    for (int i = 1; i <= Q; i++) cin >> L[i] >> R[i];

    // アタリの個数・ハズレの個数の累積和を求める
    Atari[0] = 0;
    Hazre[0] = 0;
    for (int i = 1; i <= N; i++) {
        Atari[i] = Atari[i - 1]; if (A[i] == 1) Atari[i] += 1;
        Hazre[i] = Hazre[i - 1]; if (A[i] == 0) Hazre[i] += 1;
    }

    // 質問に答える
    for (int i = 1; i <= Q; i++) {
        int NumAtari = Atari[R[i]] - Atari[L[i] - 1];
        int NumHazre = Hazre[R[i]] - Hazre[L[i] - 1];
        if (NumAtari > NumHazre) cout << "win" << endl;
        else if (NumAtari == NumHazre) cout << "draw" << endl;
        else cout << "lose" << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

2.2 問題 B07：Convenience Store 2 (難易度：★3相当)

この問題では、$t = 0,1,\ldots,T-1$ について「時刻 $t+0.5$ には何人働いているのか Answer[t]」を求める必要があります。

それでは、Answer[t] の値はどうやって計算すれば良いのでしょうか。時刻 L から時刻 R まで働く人については、$t = L, L+1, \ldots, R-1$ について「時刻 $t+0.5$ の労働者数」を 1 だけ増やすので、以下のプログラムによって正しく計算することができます。

```
// 入力
cin >> T >> N;
for (int i = 1; i <= N; i++) cin >> L[i] >> R[i];

// 答えを求める
for (int i = 0; i < T i++) Answer[i] = 0;
for (int i = 1; i <= N; i++) {
    for (int j = L[i]; j < R[i]; j++) Answer[j] += 1;
}

// 出力
for (int d = 0; d < T; d++) cout << Answer[d] << endl;
```

しかし、このプログラムの計算量は $O(NT)$ です。本問題の制約は $N, T \leq 10^5$ であるため、残念ながら実行時間制限に間に合いません。

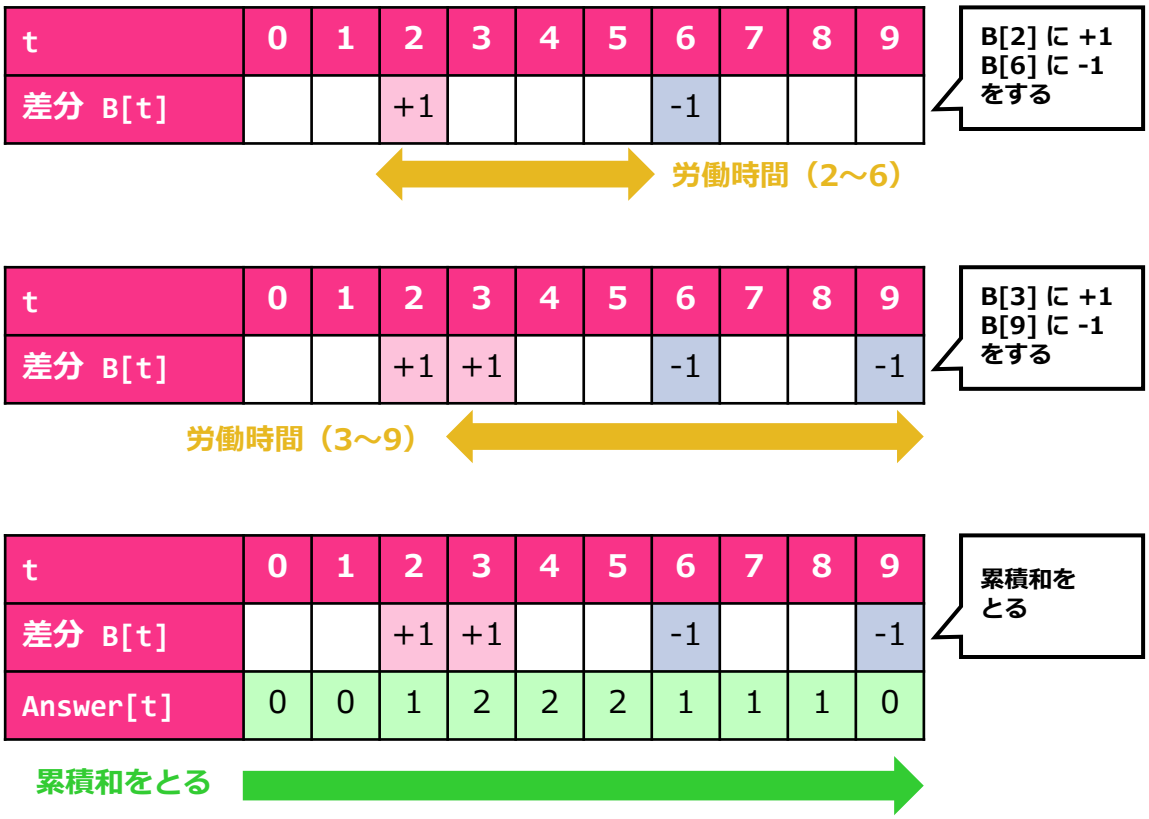
差分を計算しよう

そこで、各時刻の労働者数 Answer[t] の代わりに、**労働者数の前の時刻との差分 B[t]** を計算することを考えます。時刻 L から時刻 R まで働く人については、B[L] に +1 して B[R] に -1 すれば良いです。

t	0	1	2	3	4	5	6	7
(労働者数)			+1	+1	+1	+1		
差分 B[t]			+1				-1	

労働時間（2〜6）

すると、B[t] の累積和が求めるべき答え Answer[t] になります。たとえば $N = 2, T = 10, (L, R) = (2, 6), (3, 9)$ の場合は下図のように計算できます。



したがって、この問題を高速に解くプログラムは、以下の解答例のように実装することができます。計算量は $O(N + T)$ であり、実行時間制限には余裕を持って間に合います。

解答例（C++）

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

int N, T;
int L[500009], R[500009];
int Answer[500009], B[500009];

int main() {
        // 入力
        cin >> T >> N;
        for (int i = 1; i <= N; i++) cin >> L[i] >> R[i];

        // 前日比に加算
        for (int i = 0; i <= T; i++) B[i] = 0;
```

```
    for (int i = 1; i <= N; i++) {
        B[L[i]] += 1;
        B[R[i]] -= 1;
    }

    // 累積和をとる
    Answer[0] = B[0];
    for (int d = 1; d <= T; d++) Answer[d] = Answer[d - 1] + B[d];

    // 出力
    for (int d = 0; d < T; d++) cout << Answer[d] << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

2.3 問題 B08：Counting Points (難易度：★4相当)

まず考えられる解法は、それぞれの点について「x 座標が a 以上 c 以下であり、y 座標が b 以上 d 以下であるかどうか」を直接調べることです。

しかしこの解法では、1 つの質問に答えるのに計算量 $O(N)$ かかってしまいます。質問の個数は Q 個なので、全体の計算量は $O(NQ)$ となり、残念ながら実行時間制限に間に合いません。

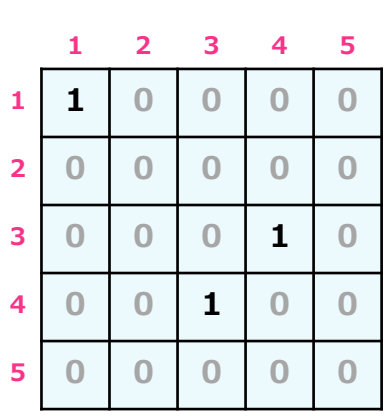
二次元累積和を考えよう

本問題では点の座標 X_i, Y_i が 1 以上 1500 以下の整数となっているので、以下のような配列 `S[i][j]`（大きさ約 1500×1500）を用意します。

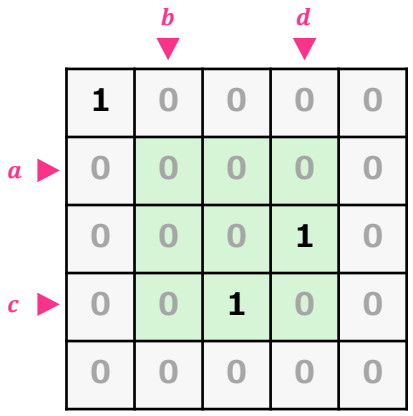
> `S[i][j]`：座標 (i, j) には何個の点が存在するか？

たとえば、点が座標 $(1,1), (3,4), (4,3)$ に存在する場合、配列 `S[i][j]` は下図左側のようになります。

そこで、質問の答えである「x 座標が a 以上 c 以下であり、y 座標が b 以上 d 以下である点の個数」は、下図右側のような長方形領域の総和となるため、**二次元累積和**を使って計算することができます。



配列 `S[i][j]` の値



答えはどの部分の総和か？

具体的には、配列 S[i][j] の二次元累積和を T[i][j] とするとき、質問の答えは次の式で表すことができます。

```
T[c][d] + T[a-1][b-1] - T[a-1][d] - T[c][b-1]
```

したがって、以下の解答例のようなプログラムにより、計算量 $O(1)$ で各質問に答えることができます。

解答例（C++）

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

// 入力で与えられる変数
int N, X[100009], Y[100009];
int Q, A[100009], B[100009], C[100009], D[100009];

// 各座標にある点の数 S[i][j]、二次元累積和 T[i][j]
int S[1509][1509];
int T[1509][1509];

int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> X[i] >> Y[i];
    cin >> Q;
    for (int i = 1; i <= Q; i++) cin >> A[i] >> B[i] >> C[i] >> D[i];

    // 各座標にある点の数を数える
    for (int i = 1; i <= N; i++) S[X[i]][Y[i]] += 1;

    // 累積和をとる
    for (int i = 0; i <= 1500; i++) {
        for (int j = 0; j <= 1500; j++) T[i][j] = 0;
    }
    for (int i = 1; i <= 1500; i++) {
        for (int j = 1; j <= 1500; j++) T[i][j] = T[i][j - 1] + S[i][j];
    }
    for (int i = 1; i <= 1500; i++) {
        for (int j = 1; j <= 1500; j++) T[i][j] = T[i - 1][j] + T[i][j];
    }

    // 答えを求める
    for (int i = 1; i <= Q; i++) {
        cout << T[C[i]][D[i]] + T[A[i] - 1][B[i] - 1] - T[A[i] - 1][D[i]] -
T[C[i]][B[i] - 1] << endl;
    }
    return 0;
}
```

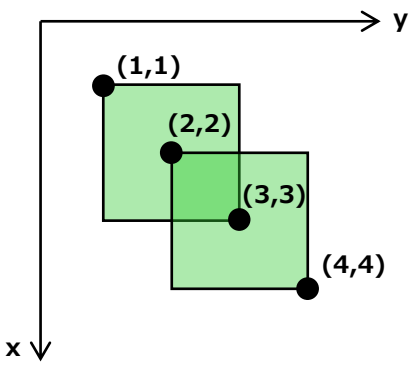
※Python のコードはサポートページをご覧ください

2.4 問題 B09：Papers (難易度：★4相当)

まずは以下の配列を考えます。T[i][j] が 1 以上になっている (i, j) の個数が、求めるべき答え（紙が 1 枚以上置かれている領域の面積）です。

T[i][j]：座標 $(i + 0.5, j + 0.5)$ には何枚の紙が置かれているか？

たとえば、隅の座標が (1, 1)・(3, 3) である紙と、隅の座標が (2, 2)・(4, 4) である紙が置かれている場合、T[i][j] の値は下図右側のようになります。



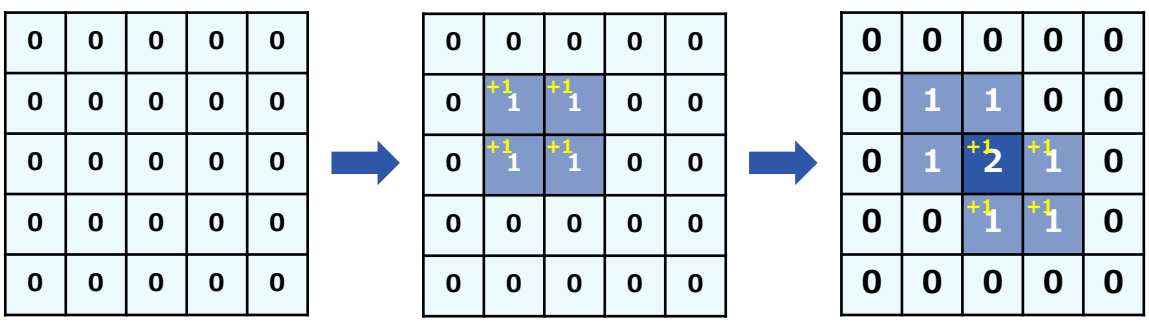
紙の置かれ方※1

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	0	0
2	0	1	2	1	0
3	0	0	1	1	0
4	0	0	0	0	0

配列 T[i][j] の値

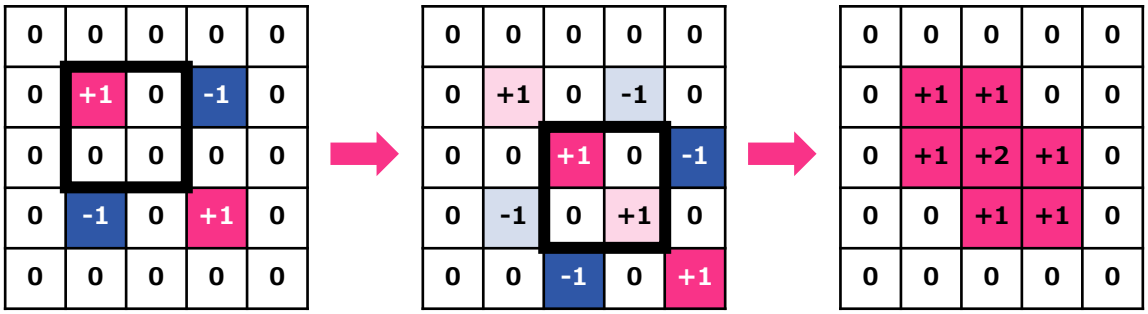
T[i][j] の計算

それでは、T[i][j] の値はどうやって計算すれば良いのでしょうか。もちろん、それぞれの紙について「対応する部分に +1 をする」という方法でも上手くいきますが、計算に時間がかかってしまいます。



※1　説明の都合上、上下方向を x 軸、左右方向を y 軸にしています

そこで 67 ページ（2.4 節）で説明したように、四隅に +1/-1 する操作を行った後、最後に二次元累積和を取ると、より高速に T[i][j] の値を計算することができます。具体例を以下に示します。



以上のアルゴリズムを実装すると、解答例のようになります。+1/-1 を加算する位置が、問題 A09 と微妙に異なることに注意してください。

たとえば問題 A09 では配列の $(c+1, d+1)$ 番目に +1 をしていますが、今回は配列の (c, d) 番目に +1 をしています。

解答例（C++）

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

// 入力で与えられる変数
int N;
int A[100009], B[100009], C[100009], D[100009];

// 座標 (i+0.5, j+0.5) に置かれている紙の数 T[i][j]
int T[1509][1509];

int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> A[i] >> B[i] >> C[i] >> D[i];

    // 各紙について +1/-1 を加算
    for (int i = 0; i <= 1500; i++) {
        for (int j = 0; j <= 1500; j++) T[i][j] = 0;
    }
    for (int i = 1; i <= N; i++) {
        T[A[i]][B[i]] += 1;
        T[A[i]][D[i]] -= 1;
        T[C[i]][B[i]] -= 1;
        T[C[i]][D[i]] += 1;
    }
```

```
    // 二次元累積和をとる
    for (int i = 1; i <= 1500; i++) {
        for (int j = 1; j <= 1500; j++) T[i][j] = T[i][j - 1] + T[i][j];
    }
    for (int i = 1; i <= 1500; i++) {
        for (int j = 1; j <= 1500; j++) T[i][j] = T[i - 1][j] + T[i][j];
    }

    // 面積を数える
    int Answer = 0;
    for (int i = 0; i <= 1500; i++) {
        for (int j = 0; j <= 1500; j++) {
                if (T[i][j] >= 1) Answer += 1;
        }
    }
    cout << Answer << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください