進階線段樹

日月卦長

1. 假的區間修改

經典題

- 給你一個長度為 n 的陣列 a，再給你 q 個操作，操作有兩種：
- $query(ql, qr)$:
  查詢 $a_{ql} + a_{ql+1} + \cdots + a_{qr}$ 的值
- $update(ql, qr)$:
  $\forall ql \le p \le qr$, 將 $a_p = \left\lfloor \sqrt{a_p} \right\rfloor$
- $1 \le n, q, a_i \le 10^6$

懶惰標記怎麼設?

a

0	1	2	3	4	5	6	7
1	9	6	5	5	4	1	3

$update(0, 4)$

a

0	1	2	3	4	5	6	7
1	3	2	2	2	4	1	3

開根號的次數

- 若 $x > 1$ ， 將 x 連續開根號 $O(\log \log x)$ 次後就會變成 1

$$7122 \to 84 \to 9 \to 3 \to 1$$

- 10^6 內的數字最多開 5 次根號就會變成 1

直接暴力一個一個改？

整個區間都是 1 就不要改

- 線段樹的節點要記錄區間最大值

```
struct Node {
  int Max, Sum;
  Node(int val) : Max(Val), Sum(Val) {}
  Node operator+(Node Other) const {
    Other.Max = max(Max, Other.Max);
    Other.Sum += Sum;
    return Other;
  }
};
```

整個區間都是 1 就不要改

- 修改的時候只改不是 1 的區域

```
vector<Node> Tree;
void update(int ql, int qr, int l, int r, int d) {
  if (r < ql || qr < l)
    return; // 不再範圍內
  if (Tree[d].Max <= 1)
    return; // 整個區間都已經是 1 了
  if (l == r) {
    Tree[d].Max = Tree[d].Sum = sqrtl(Tree[d].Sum);
  } else {
    int mid = (l + r) / 2;
    update(ql, qr, l, mid, d * 2);
    update(ql, qr, mid + 1, r, d * 2 + 1);
    Tree[d] = Tree[d * 2] + Tree[d * 2 + 1];
  }
}
```

2. 二分搜

CSES Hotel Queries

- 有 n 間旅館，編號 $1\sim n$，第 i 間能容納 h_i 個人
  接著依序來了 m 組旅行團，第 i 組的人數為 t_i
- 旅行團會住進可以容納他們所有人的旅館中編號最小的那間
- 輸出每個旅行團會住進的旅館編號，如果住不進去就輸出 0
- $1 \leq n, m \leq 2 \times 10^5$

想法：二分搜前綴最大值

h

1	2	3	4	5	6	7	8
3	2	4	1	5	5	2	6

每個前綴
計算最大值

有 4 人想入住

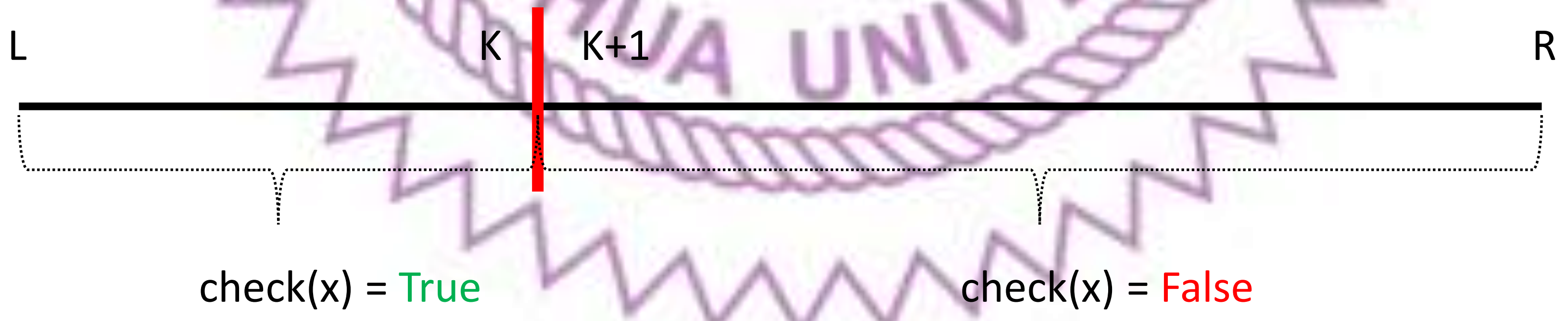
< 4 ≥ 4

h^*

1	2	3	4	5	6	7	8
3	3	4	4	5	5	5	6

複習：模板化的二分搜

```
template <class Ty, class FuncTy>
pair<Ty, Ty> binarySearch(Ty L, Ty R, FuncTy check) {
  if (check(R) == true) return {R, R + 1};
  if (check(L) == false) return {L - 1, L};
  while (L + 1 < R) {
    Ty Mid = L + (R - L) / 2;
    if (check(Mid)) L = Mid;
    else R = Mid;
  }
  return {L, R};
}
```



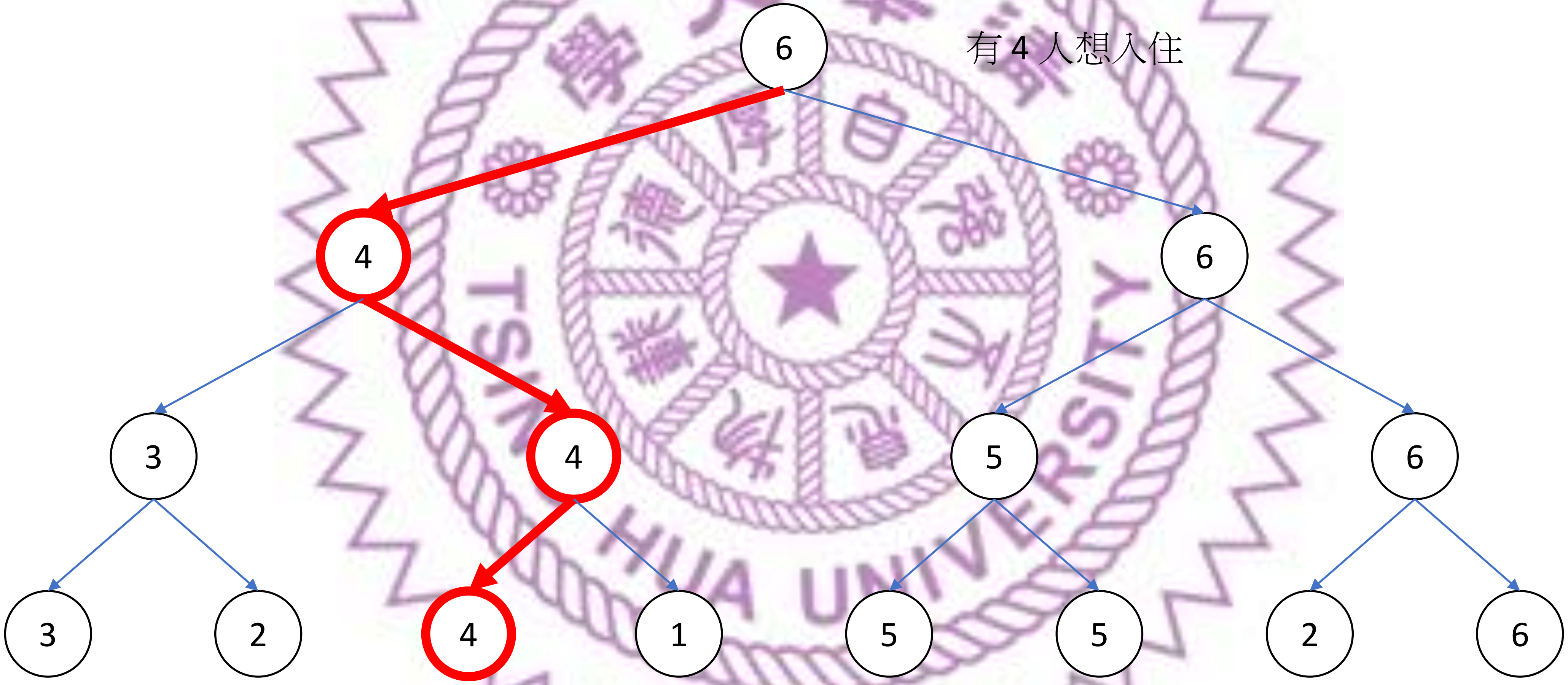
二分搜答案

- 假設有線段樹 ST:
- ST.update(p, Val):
  - 將 $h_p = Val$
- ST.query(ql,qr):
  - 計算 $\max_{ql \le i \le qr} \{h_i\}$

```
int checkin(int people_num) {
  int ans = binarySearch(1, n, [&](int Idx) {
              return ST.query(1, Idx) < people_num;
            }).second;
  if (ans > n)
    return 0;
  ST.update(ans, h[ans] -= people_num);
  return ans;
}
```

$O(\log n \log n)$

線段樹上直接二分搜 $O(\log n)$



線段樹上直接二分搜 $O(\log n)$

```
int queryWithUpdate(int val, int l, int r, int d = 1) {
  if (l == r) {
    Tree[d] -= val;
    return l;
  } else {
    int mid = (l + r) / 2;
    int ans = 0;
    if (Tree[d * 2] >= val)
      ans = query(val, l, mid, d * 2);
    else
      ans = query(val, mid + 1, r, d * 2 + 1);
    Tree[d] = max(Tree[d * 2], Tree[d * 2 + 1]);
    return ans;
  }
}
```

3. 結合律

Yosupo - Point Set Range Composite

- 給你兩個長度為 n 的陣列 a, b，再給你 q 個操作，操作有兩種：
- $query(ql, qr, x)$:
  設 $f_i(x) = a_i x + b_i$，計算 $f_{qr-1}\left(f_{qr-2}\left(\ldots\left(f_{ql}(x)\right)\right)\right) \bmod 998244353$
- $update(p, c, d)$:
  將 $a_p = c, b_p = d$
- $1 \le n, q \le 5 \times 10^5$

觀察

- 設 $F_A(x) = f_2\big(f_1(x)\big) = (a_2a_1)x + (a_2b_1 + b_2)$
- 設 $F_B(x) = f_3\big(f_2(x)\big) = (a_3a_2)x + (a_3b_2 + b_3)$
- 計算 $f_3\Big(f_2\big(f_1(x)\big)\Big)$

$$= f_3\big(F_A(x)\big)$$
$$= F_B\big(f_1(x)\big)$$
$$= (a_3a_2a_1)x + (a_3a_2b_1 + a_3b_2 + b_1)$$

想法

- 設 $mid = \lfloor (l+r)/2 \rfloor$
- 設 $F_L(x) = f_{mid}\left(f_{mid-1}\left(\dots\left(f_l(x)\right)\right)\right)$
- 設 $F_R(x) = f_r\left(f_{r-1}\left(\dots\left(f_{mid+1}(x)\right)\right)\right)$

- $f_r\left(f_{r-1}\left(\dots\left(f_l(x)\right)\right)\right) = F_R(F_L(x))$

結合律！

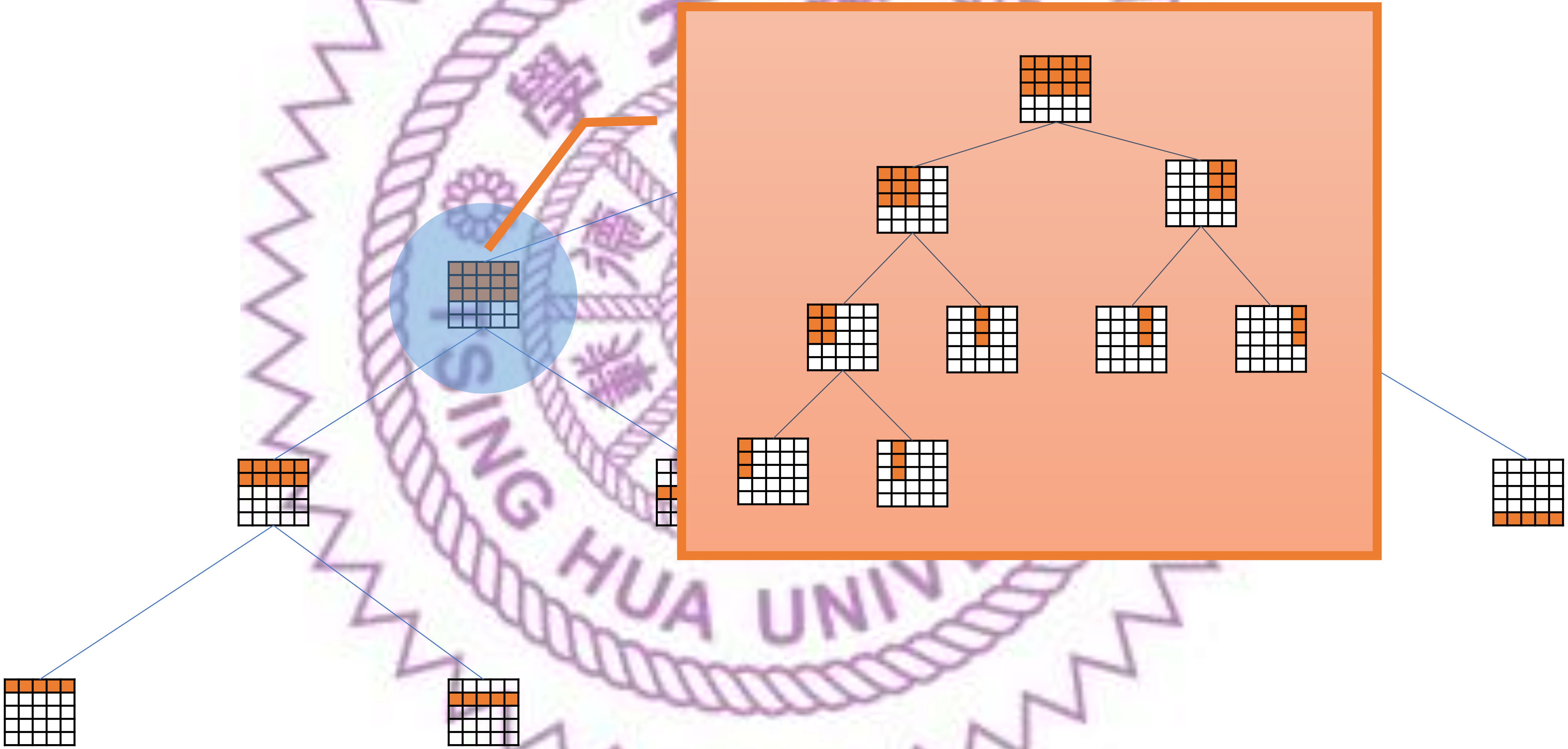
關鍵程式碼

注意 a+b != b+a (沒有交換律)

```
const long long MOD = 998244353;

struct Func {
  long long a, b;
  Func(long long a, long long b) : a(a % MOD), b(b % MOD) {}

  long long run(long long x) { return (a * x + b) % MOD; }

  Func operator+(const Func &other) const {
    long long na = a * other.a % MOD;
    long long nb = (b * other.a + other.b) % MOD;
    return Func(na, nb);
  }
};
```

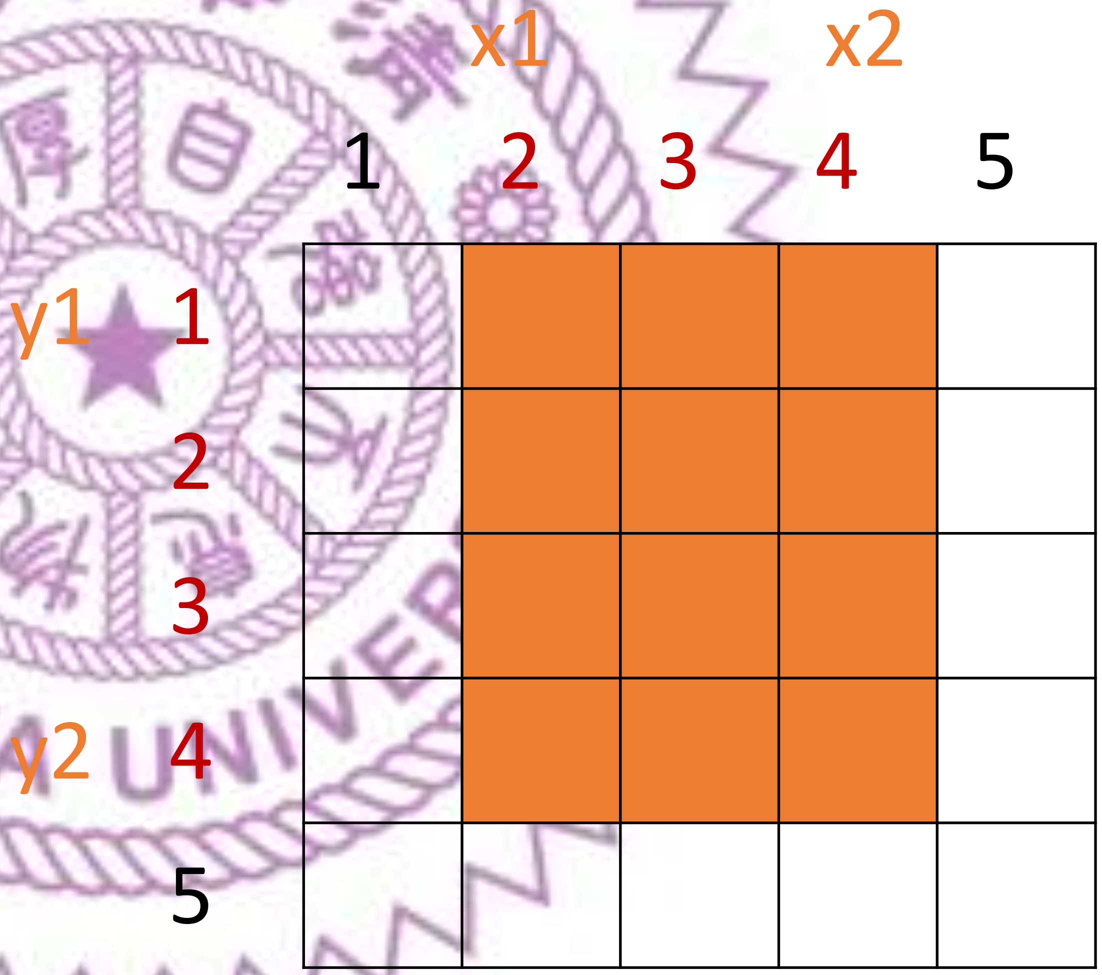
4. 更高的維度

二維線段樹

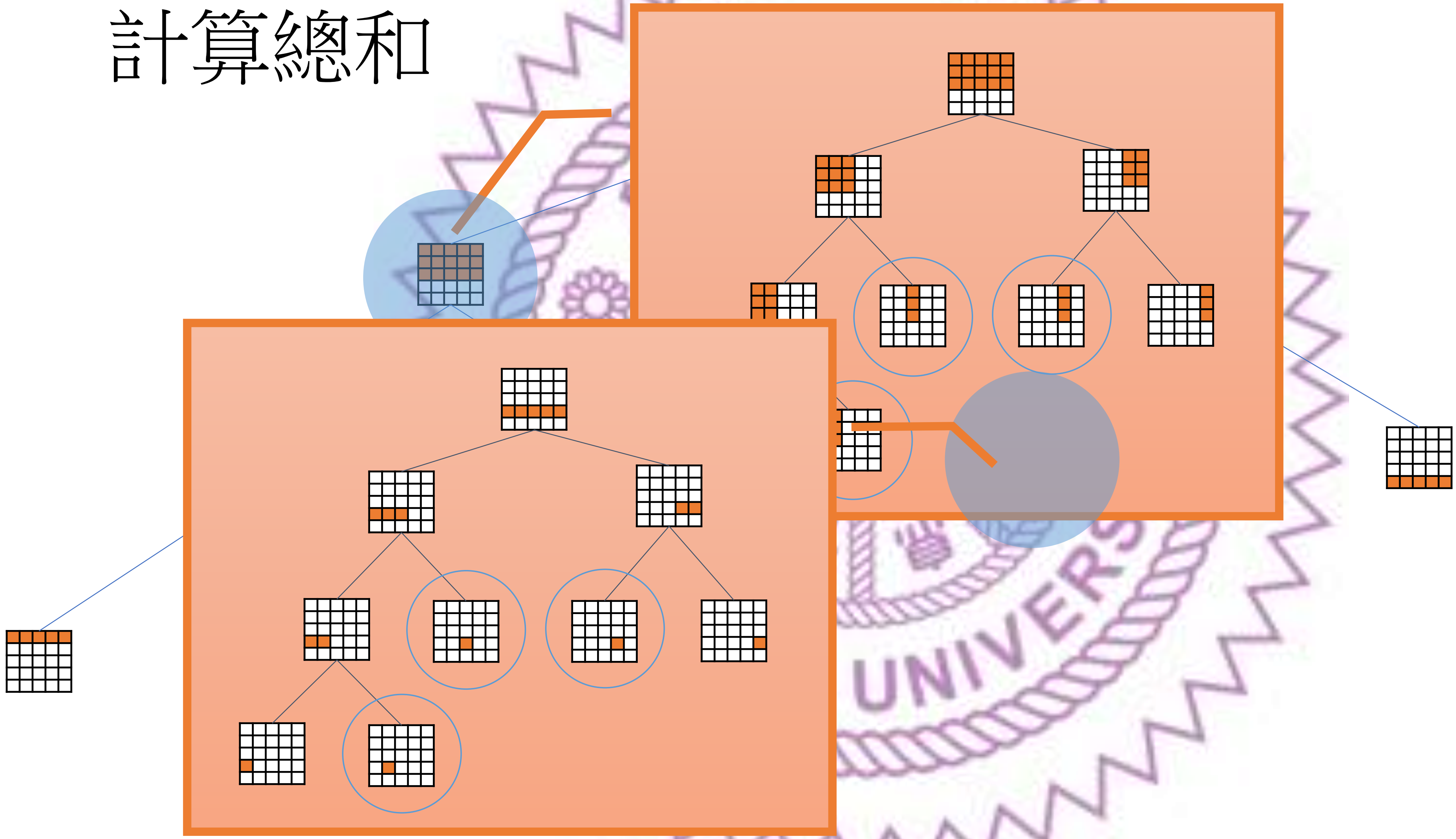


計算總和

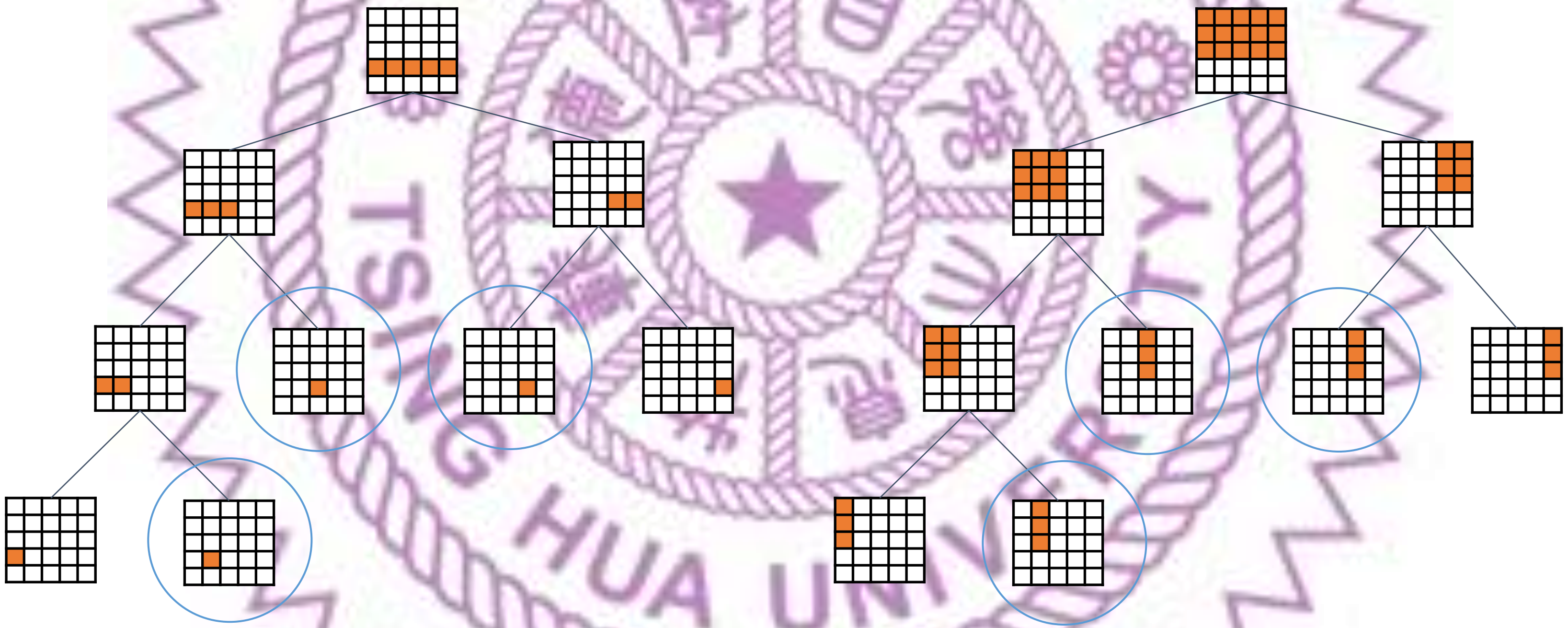
- 以計算圖色區域總和為例



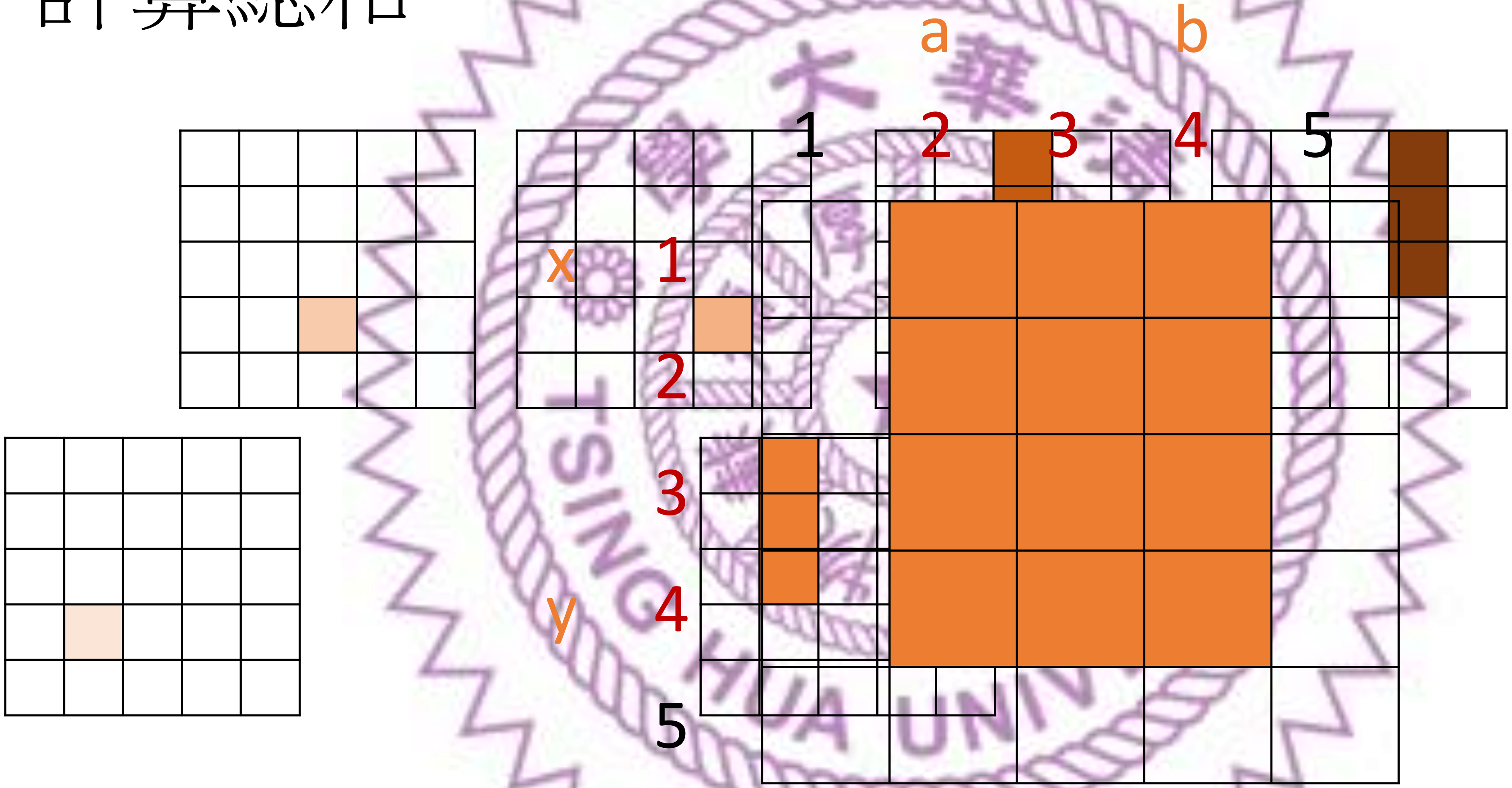
計算總和



計算總和



計算總和



懶惰標記?

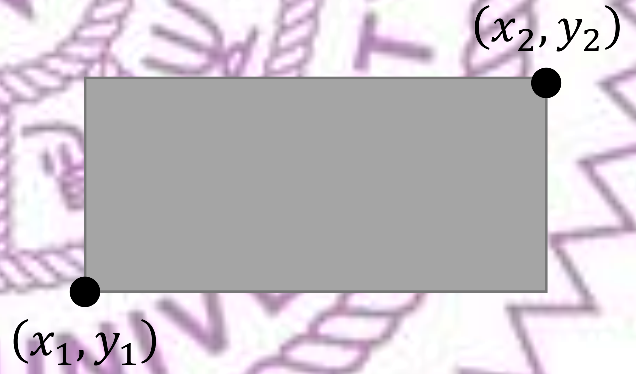
- 正常懶惰標記基本上不可能
- 只能用**永久化標記** (不做懶惰標記下推)

5. 掃描線

矩形覆蓋面積

- 平面上給你 n 個矩形，請輸出這些矩形所覆蓋的面積大小
- 每個矩形會輸入左下、右上座標 $(x_1, y_1), (x_2, y_2)$

- $1 \le n \le 10^6$
- $0 \le x_1 < x_2 \le 10^6$
- $0 \le y_1 < y_2 \le 10^6$



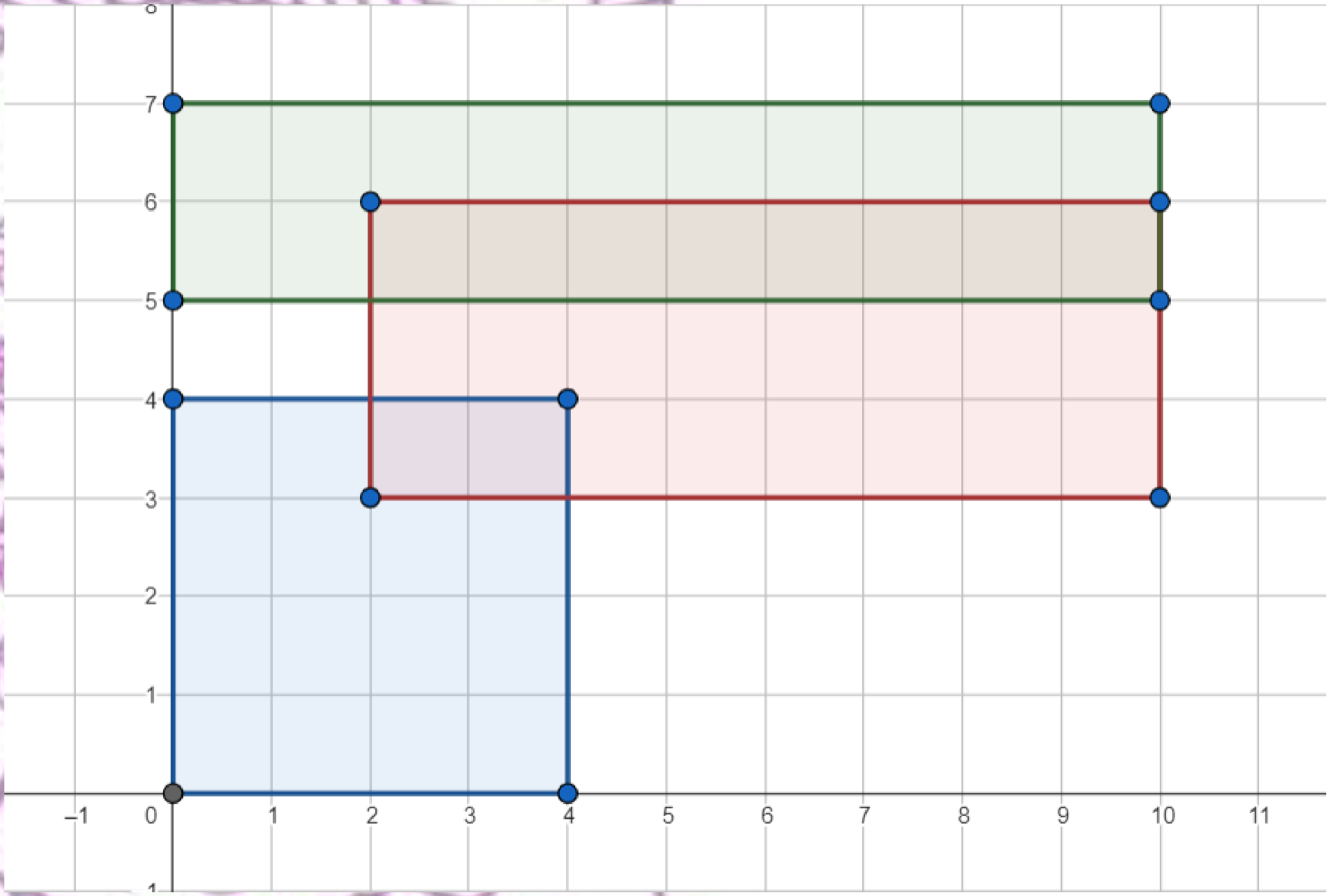
範例輸入

3

0 0 4 4

0 5 10 7

2 3 10 6

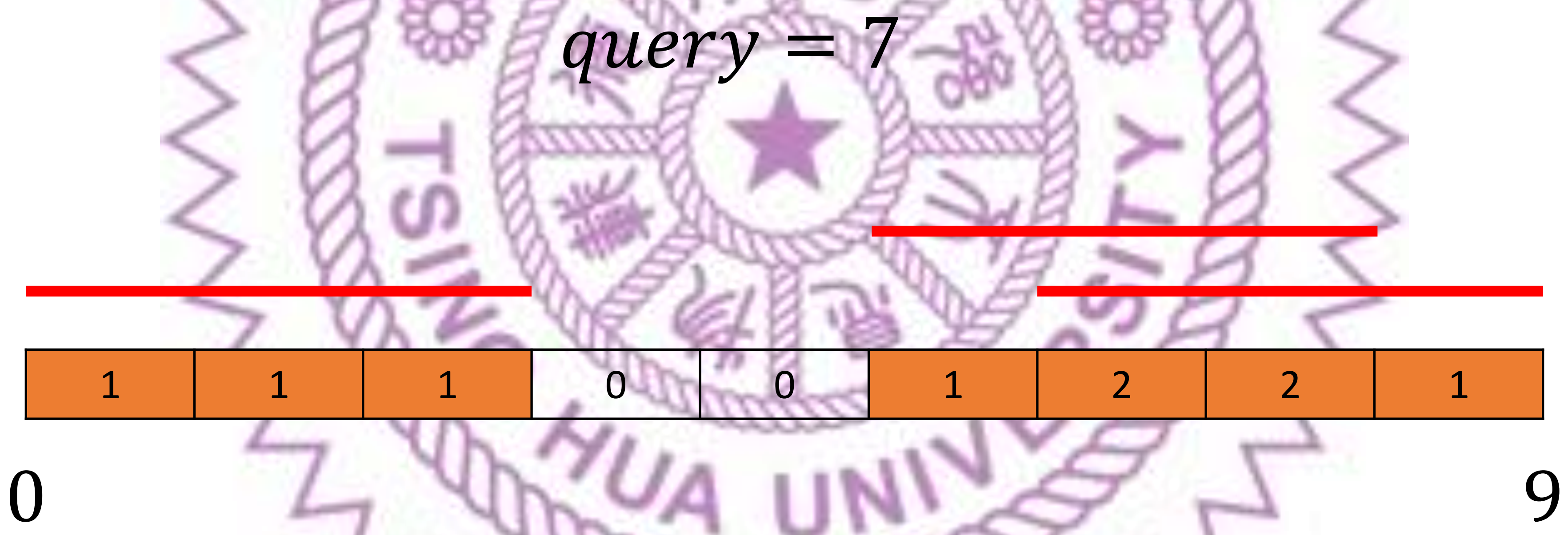


一維要能先做到動態操作

- 有一個 $[0, N]$ 的一維區間，給你 q 個操作，操作有三種：

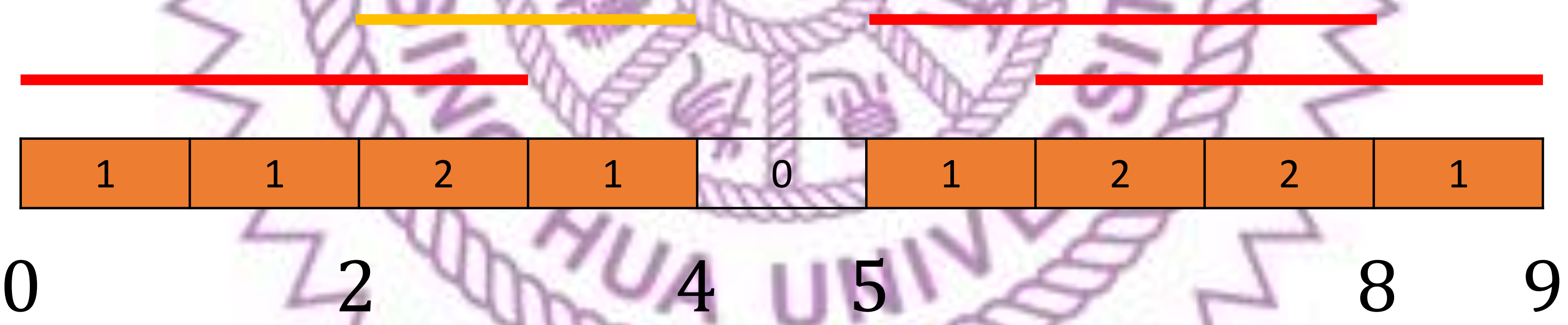
- $add(ql, qr)$:
  將一條線段覆蓋在區間 $[ql, qr]$ 上
- $remove(ql, qr)$:
  刪除剛好覆蓋在區間 $[ql, qr]$ 上的一條線段，保證線段存在
- $query$:
  查詢 $[0, N]$ 區間中，有多少區域是被線段覆蓋住的

[0,9] 區間

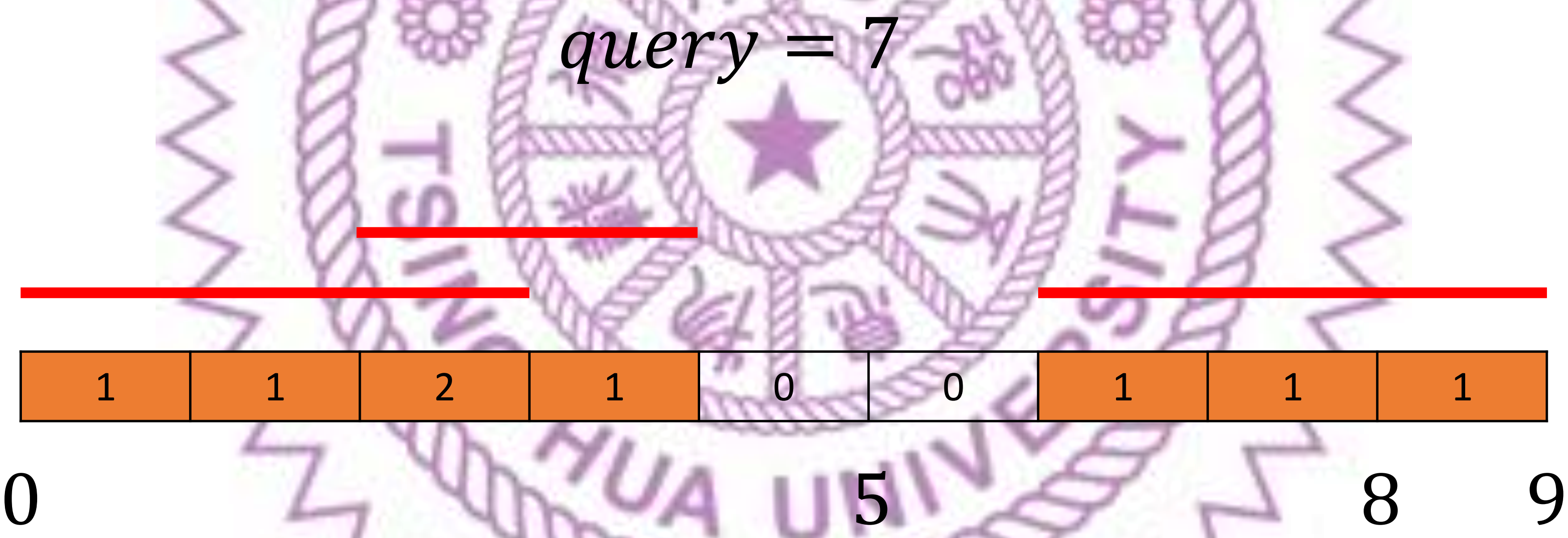


[0,9] 區間 - $add(2,4)$

$query = 8$



[0,9] 區間 - $remove(5,4)$



線段樹資訊

```cpp
struct Node {
  // 該節點的標記數量
  int tag = 0;
  // 該區間有標記的區域大小
  int sum = 0;
};

int n; // 區間範圍是 [0, n]
vector<Node> ST;

void init(int _n) {
  n = _n;
  ST.assign(n * 4, Node());
}
```

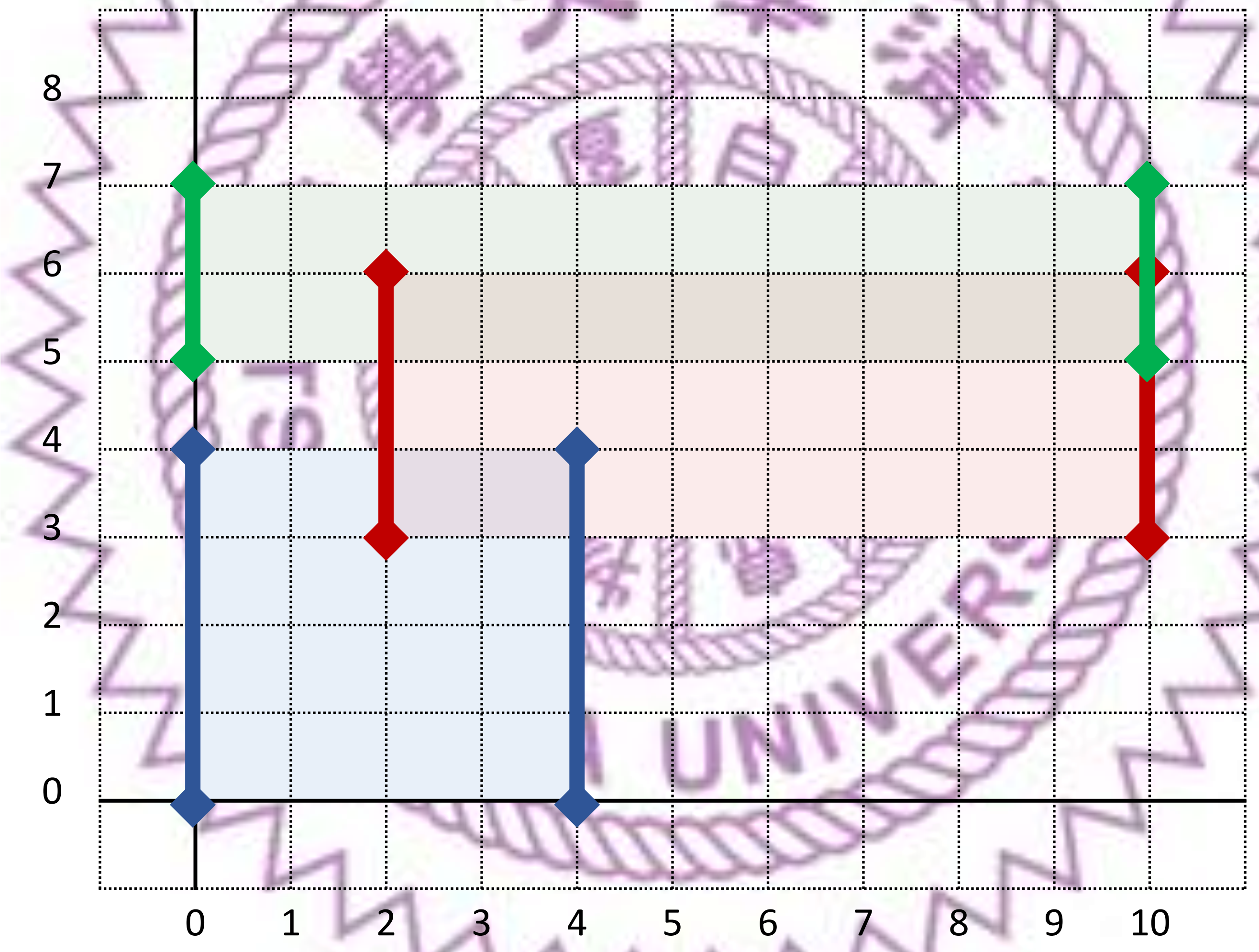
```cpp
void pull(int l, int r, int d) {
  if (ST[d].tag > 0)
    ST[d].sum = r - l;
  else if (l == r)
    ST[d].sum = 0;
  else
    ST[d].sum = ST[d * 2].sum + ST[d * 2 + 1].sum;
}
```

永久化標記更新

$add(ql, qr)$:	$update(ql, qr, 1)$
$remove(ql, qr)$:	$update(ql, qr, -1)$

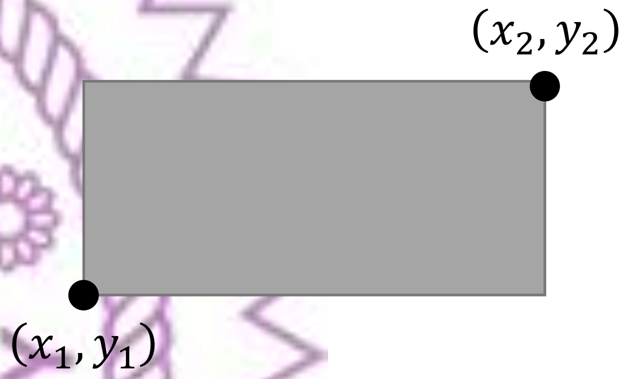
```
void update(int ql, int qr, int val, int l = 0, int r = n, int d = 1) {
  if (r <= ql || qr <= l) return; // 超過範圍
  if (ql <= l && r <= qr) { // 完全位於範圍
    ST[d].tag += val;
  } else {
    int mid = l + (r - l) / 2;
    update(ql, qr, val, l, mid, d * 2);
    update(ql, qr, val, mid, r, d * 2 + 1);
  }
  pull(l, r, d);
}
```

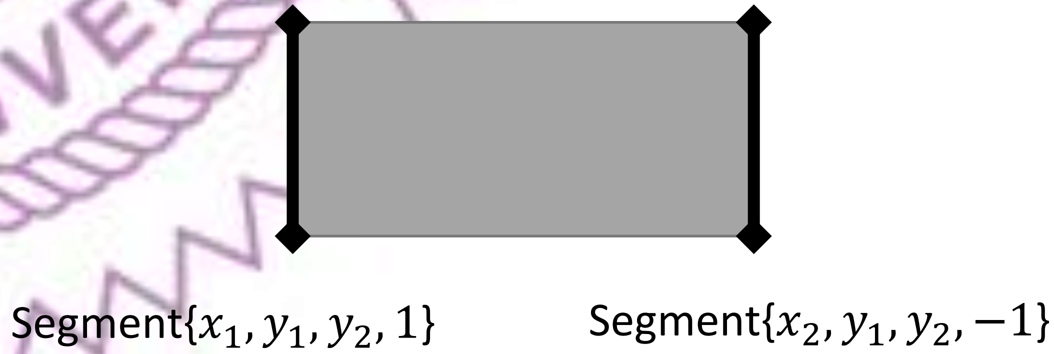
想法：將矩形拆成左右兩條垂直線段



矩形左右邊界線段資訊

```cpp
struct Segment {
  int x, y1, y2, val;
  bool operator<(const Segment &other) const {
    if (x != other.x) return x < other.x;
    return val > other.val;
  }
};
```



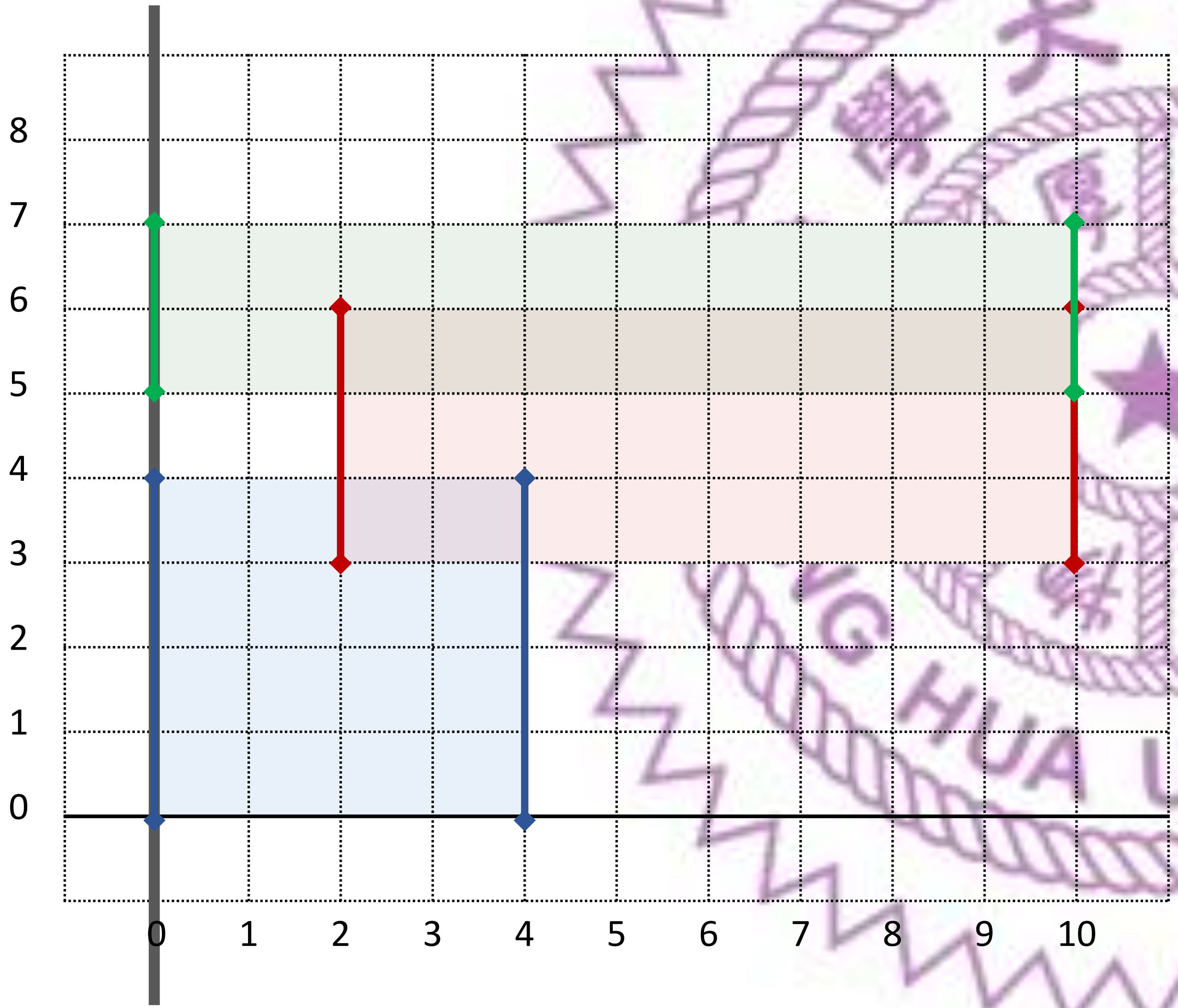


所有線段按 x 座標排序

這個用來建線段樹

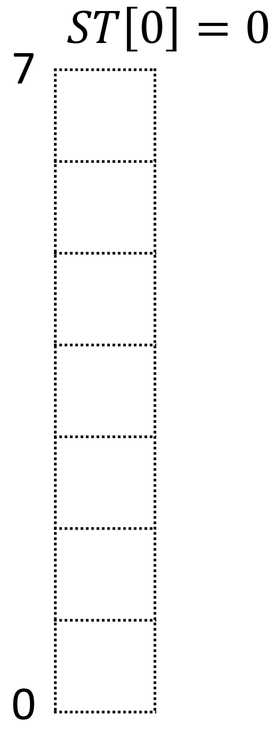
```
int N;
cin >> N;
vector&lt;Segment&gt; Segs;
int MaxY = 0;
while (N--) {
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;
  tie(y1, y2) = minmax(y1, y2);
  MaxY = max(MaxY, y2);
  Segs.emplace_back(Segment{x1, y1, y2, 1});
  Segs.emplace_back(Segment{x2, y1, y2, -1});
}
sort(Segs.begin(), Segs.end());
```

想法：用一條線從左掃到右



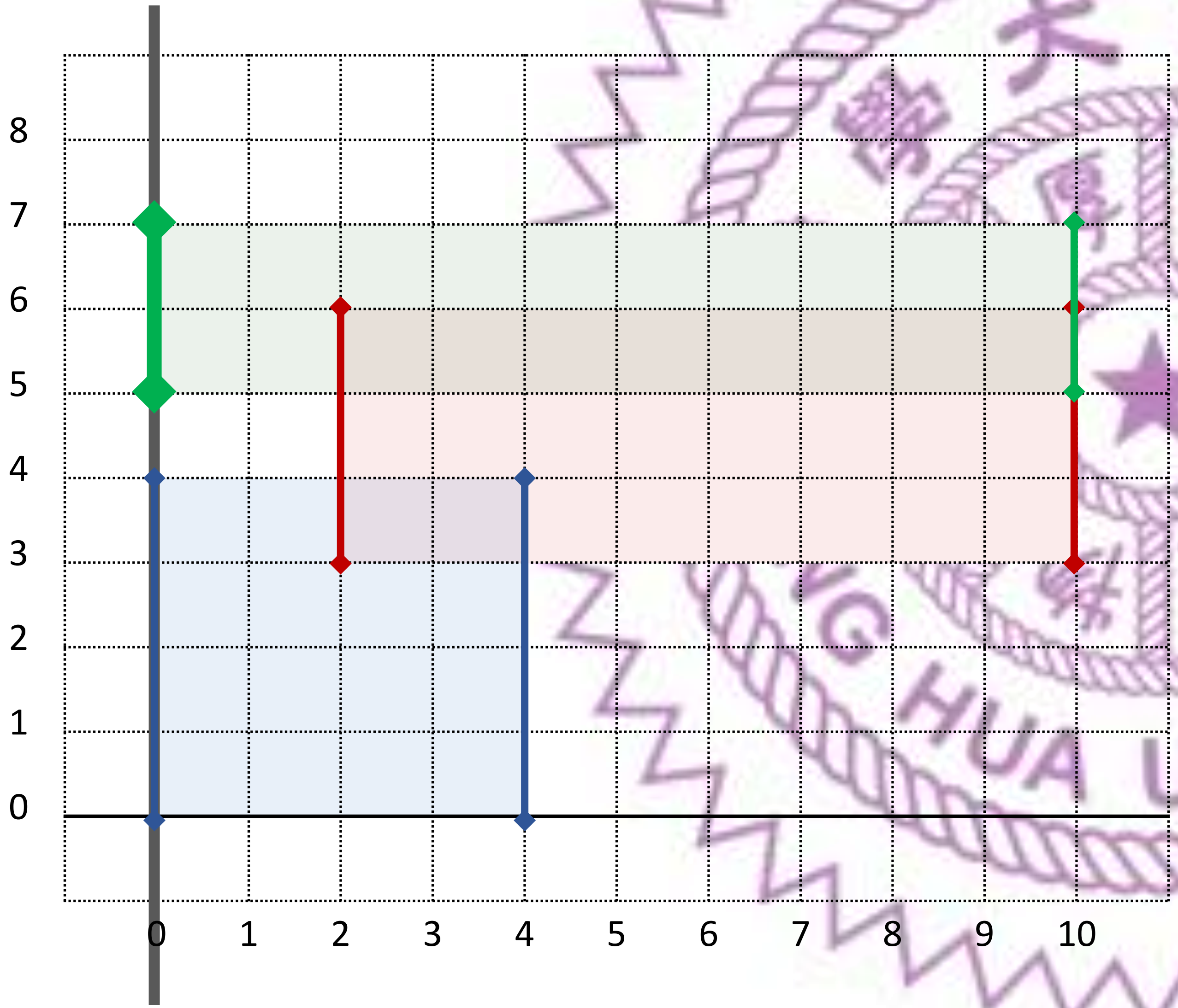
$ans = 0$

$previous\_x = 0$



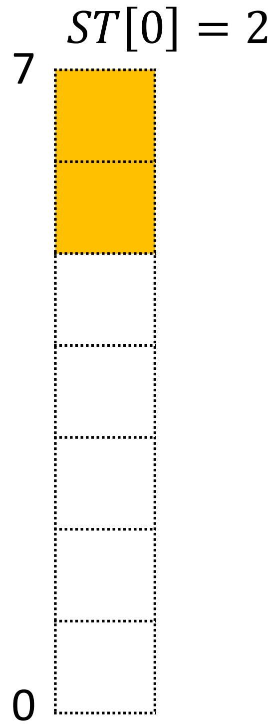

```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

想法：用一條線從左掃到右



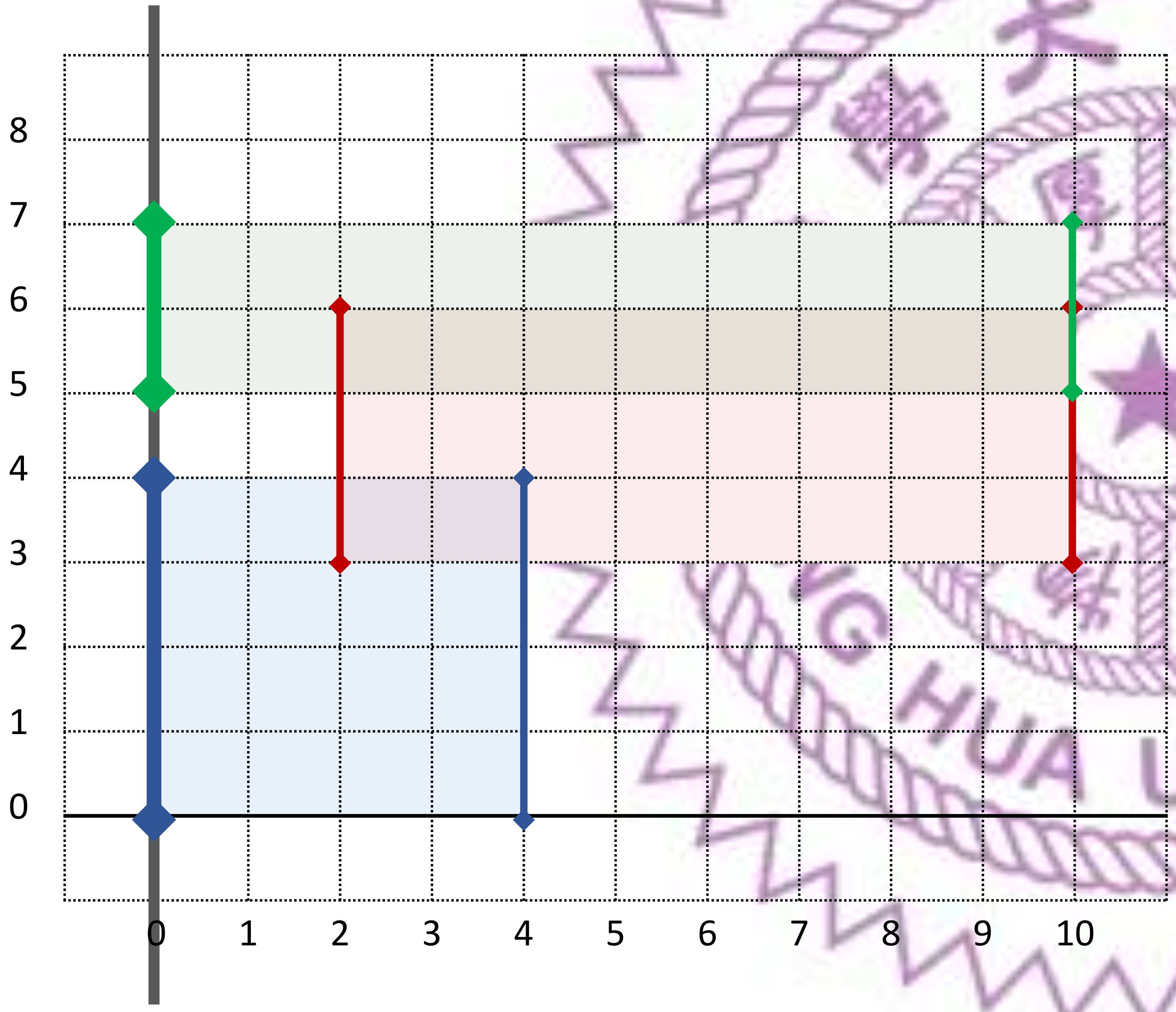
$ans = 0$
$previous\_x = 0$

$ans \mathrel{+}= (0 - 0) \times 0$
$update(5, 7, 1)$
$previous\_x = 0$



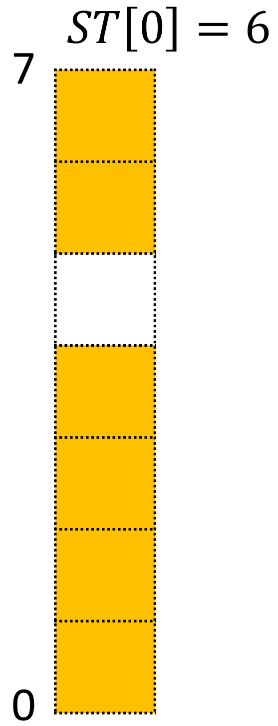

```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

想法：用一條線從左掃到右



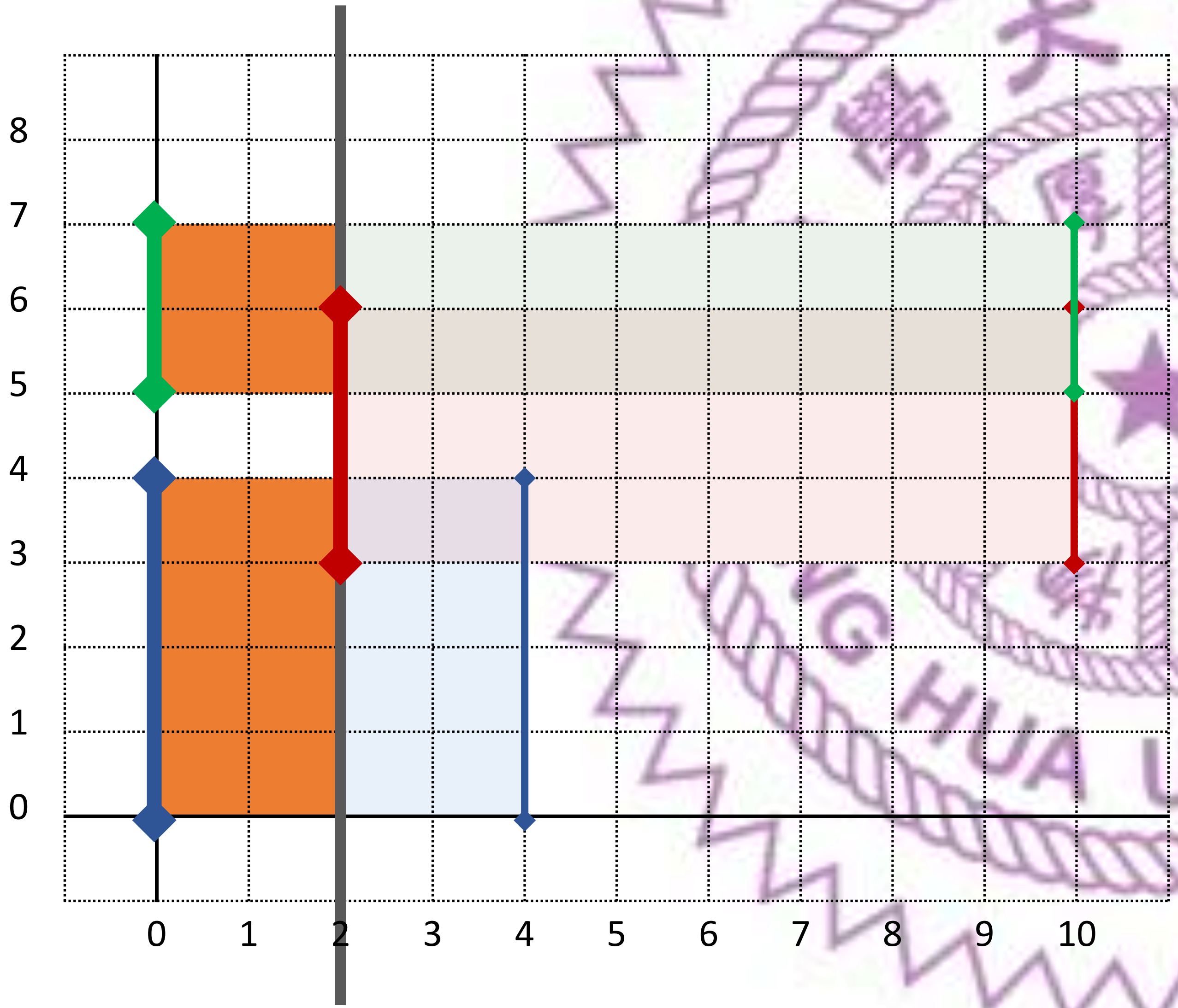
$ans = 0$
$previous\_x = 0$

$ans \mathrel{+}= (0 - 0) \times 0$
$update(0{,}4{,}1)$
$previous\_x = 0$




```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

想法：用一條線從左掃到右

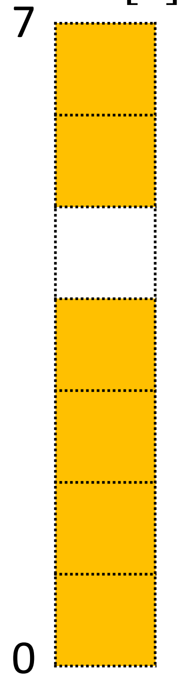


$ans = 0$

$previous\_x = 0$

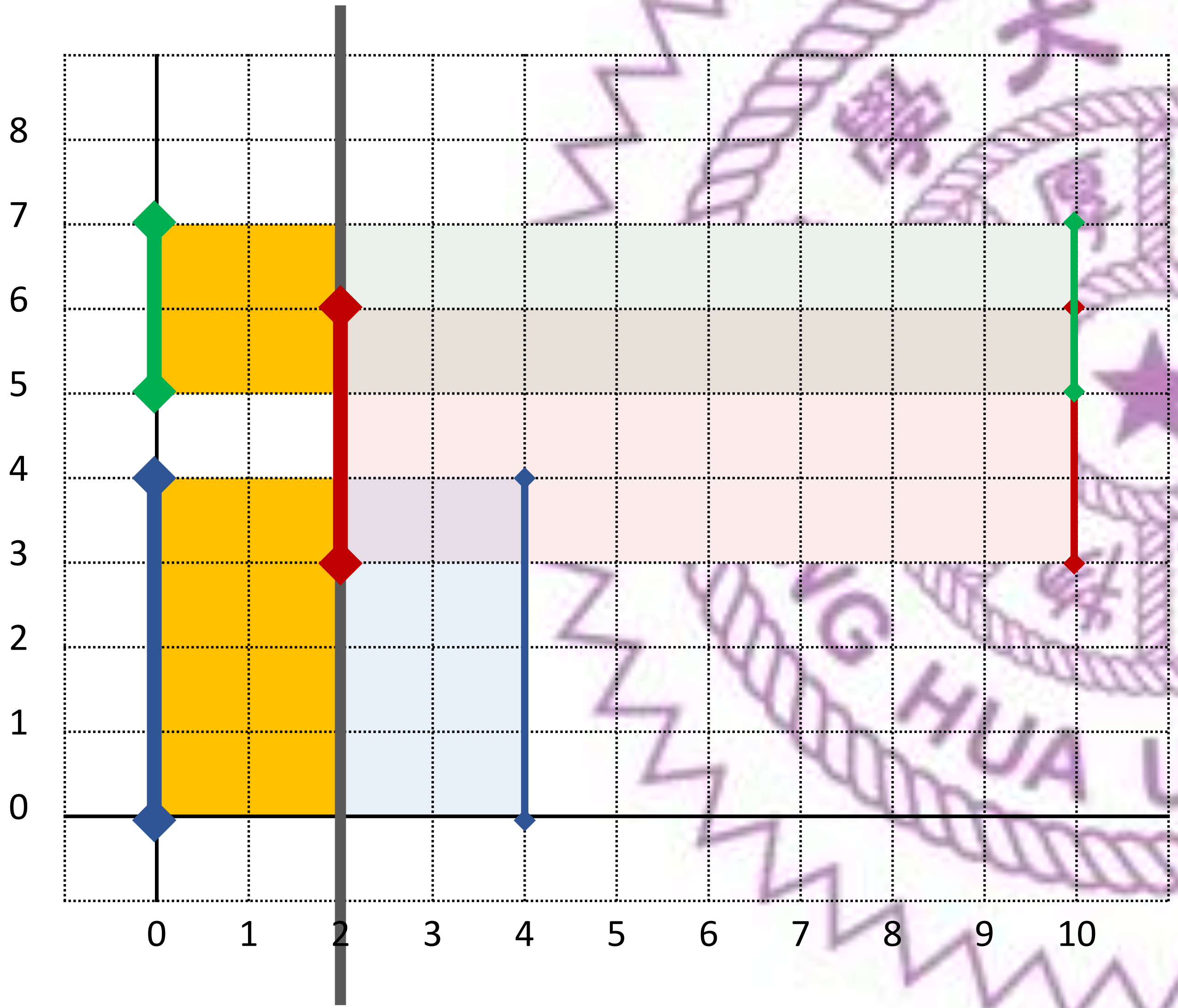
$ans \mathrel{+}= (2 - 0) \times 6$

$ST[0] = 6$




```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

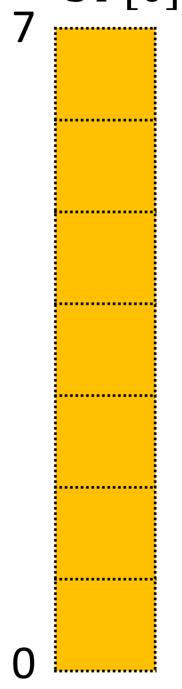
想法：用一條線從左掃到右



$ans = 12$
$previous\_x = 2$

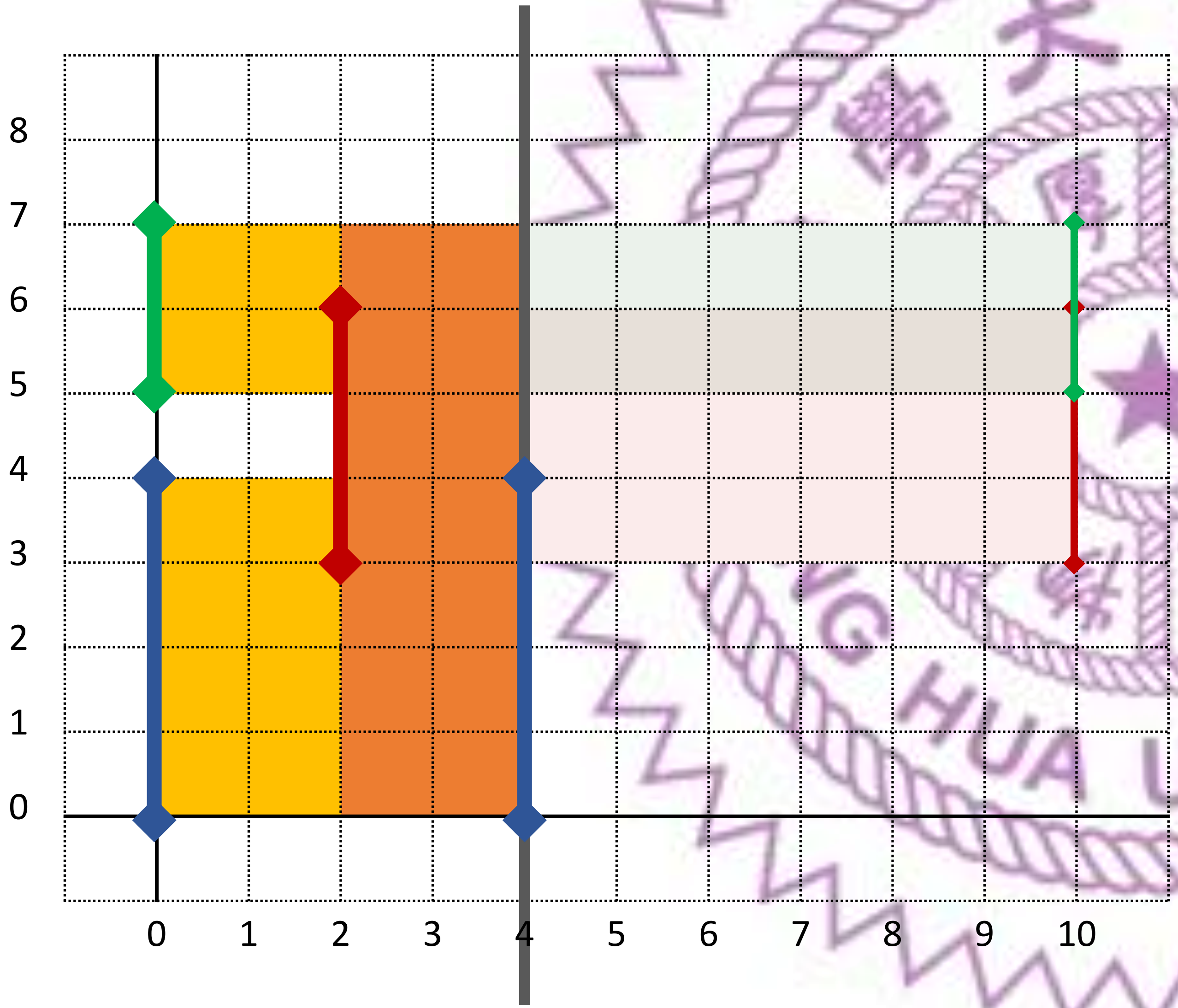
$ans \mathrel{+}= (2 - 0) \times 6$
$update(3,6,1)$
$previous\_x = 2$

$ST[0] = 7$



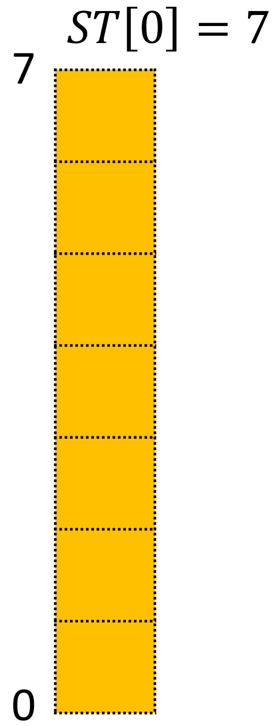

```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

想法：用一條線從左掃到右



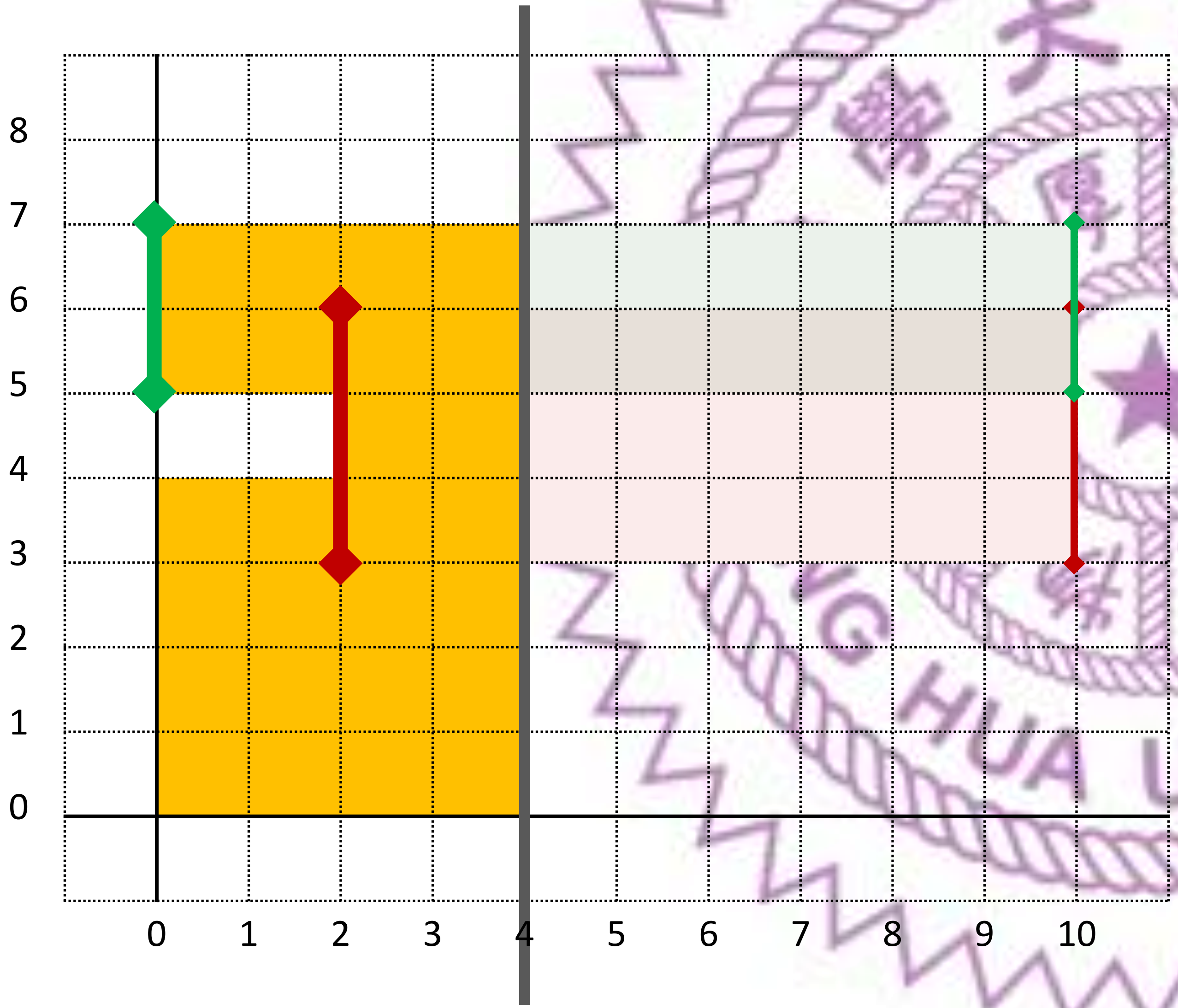
$ans = 12$
$previous\_x = 2$

$ans \mathrel{+}= (4 - 2) \times 7$



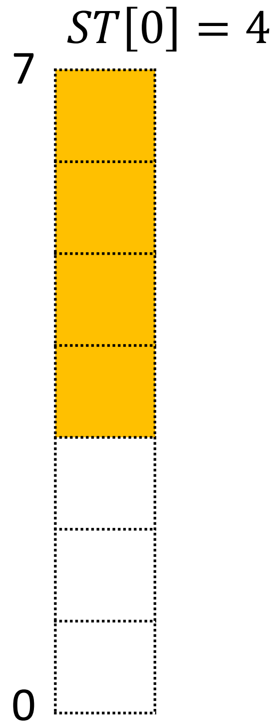

```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

想法：用一條線從左掃到右



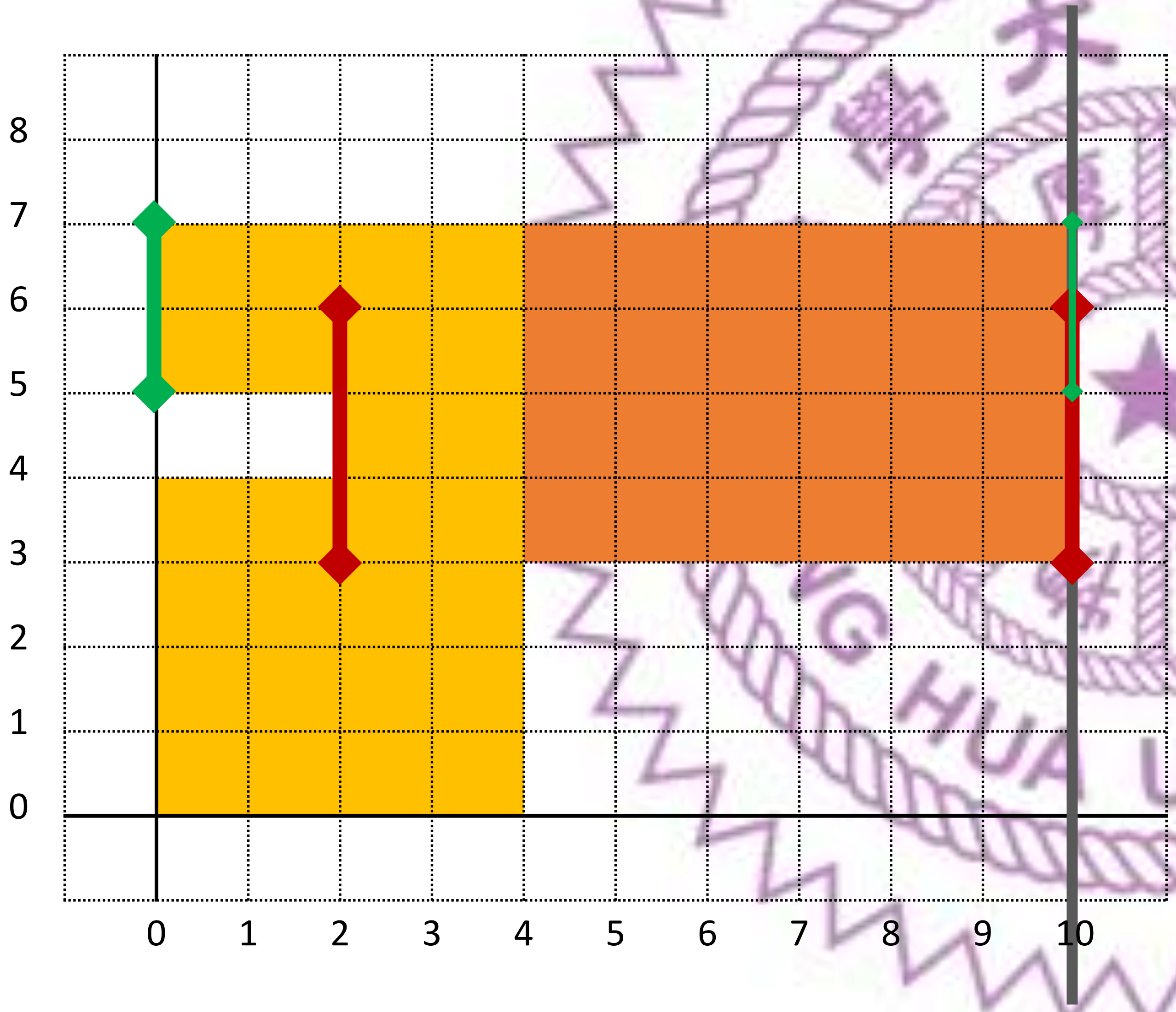
$ans = 26$
$previous\_x = 4$

$ans \mathrel{+}= (4-2) \times 7$
$update(0,4,-1)$
$previous\_x = 4$




```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

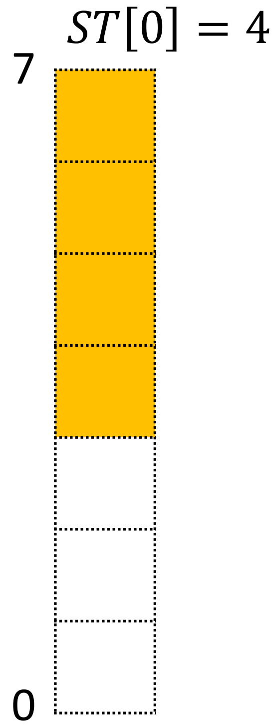
想法：用一條線從左掃到右



$ans = 26$

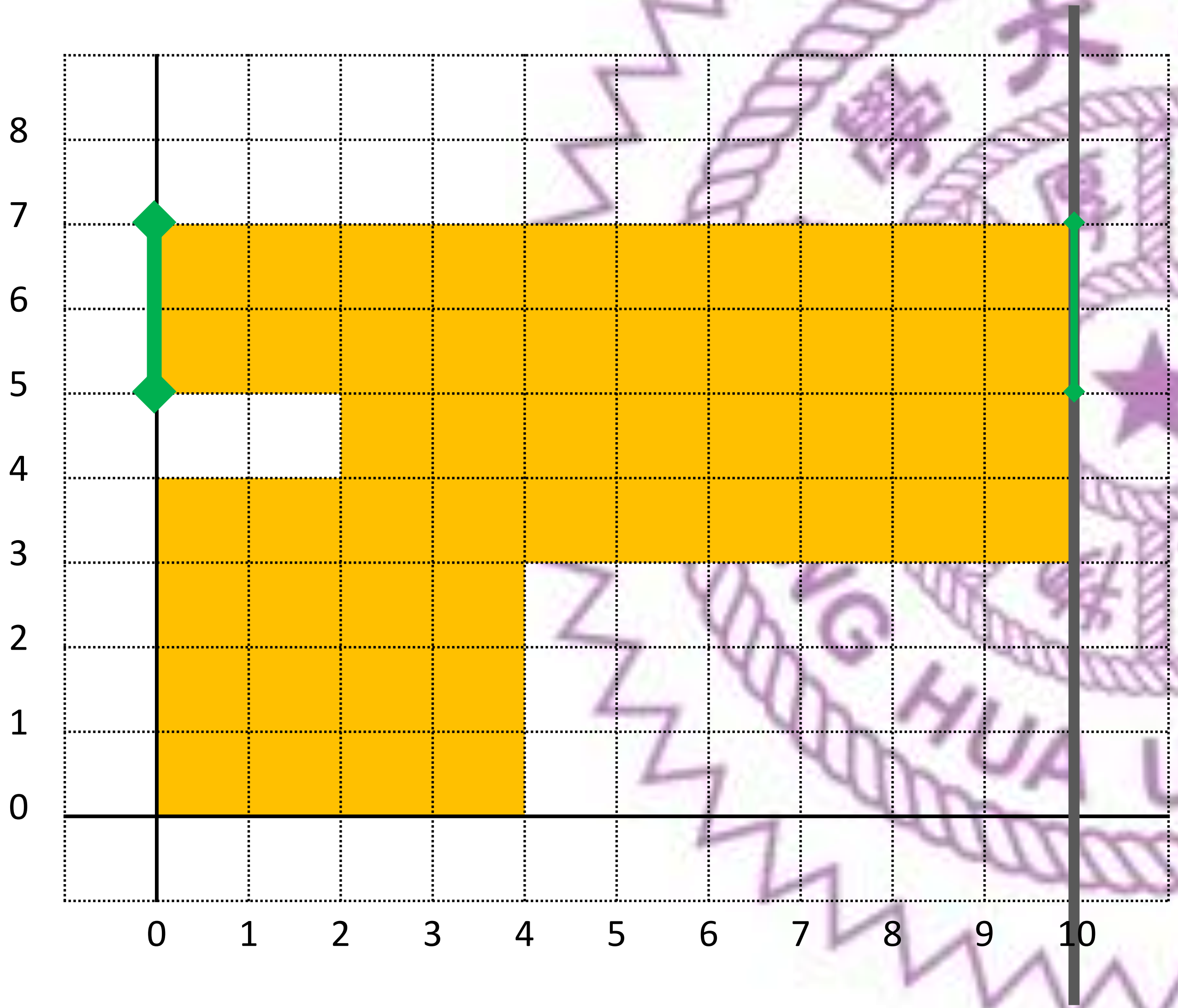
$previous\_x = 4$

$ans \mathrel{+}= (10 - 4) \times 4$




```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

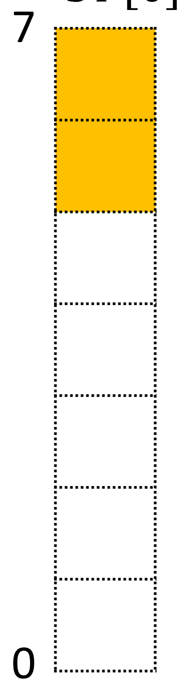
想法：用一條線從左掃到右



$ans = 50$
$previous\_x = 10$

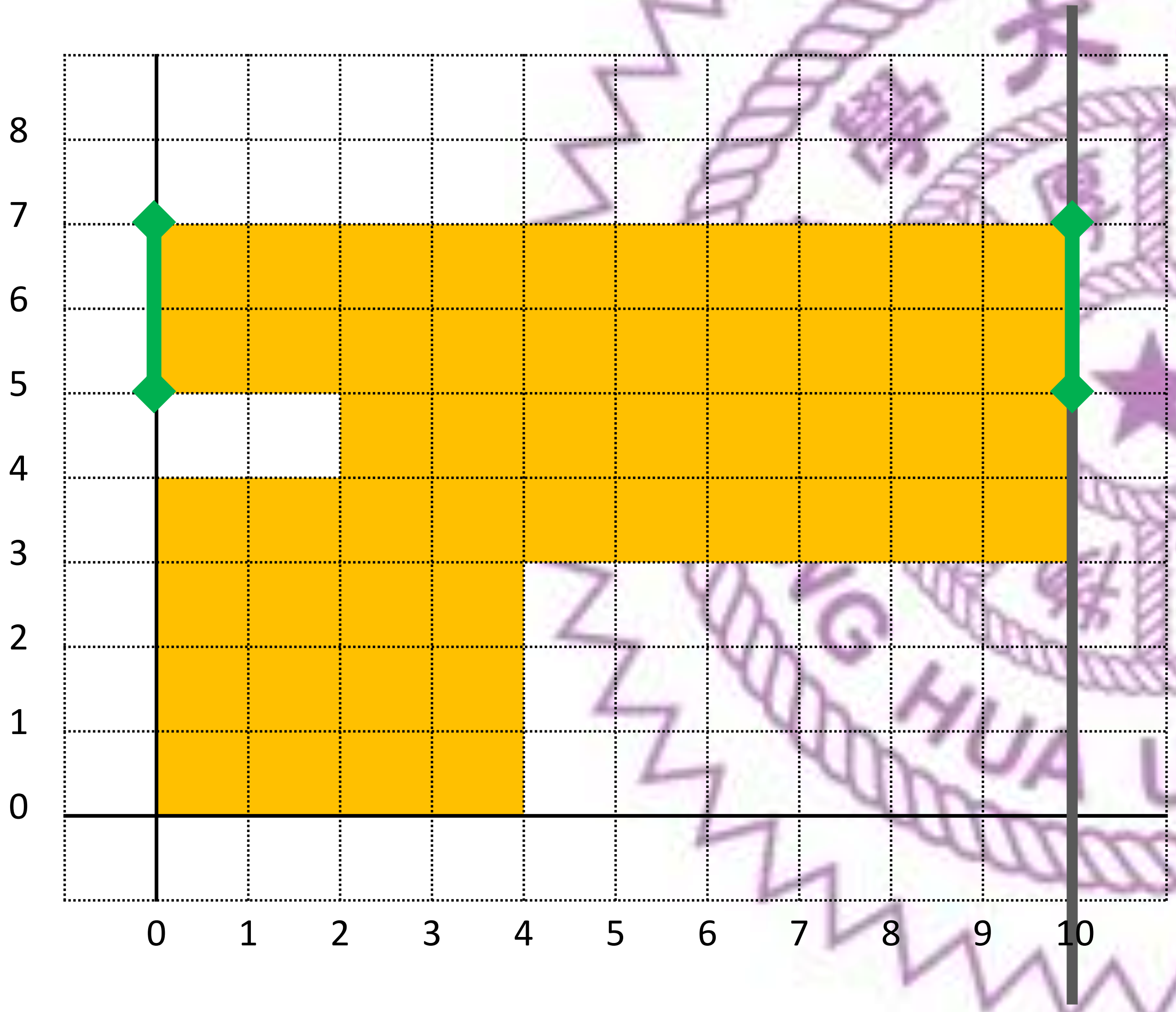
$ans \mathrel{+}= (10 - 4) \times 4$
$update(3,6,-1)$
$previous\_x = 10$

$ST[0] = 2$



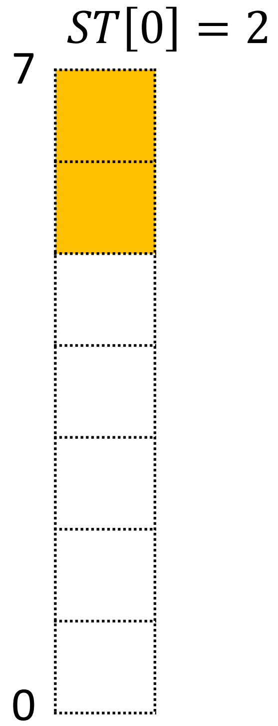

```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

想法：用一條線從左掃到右



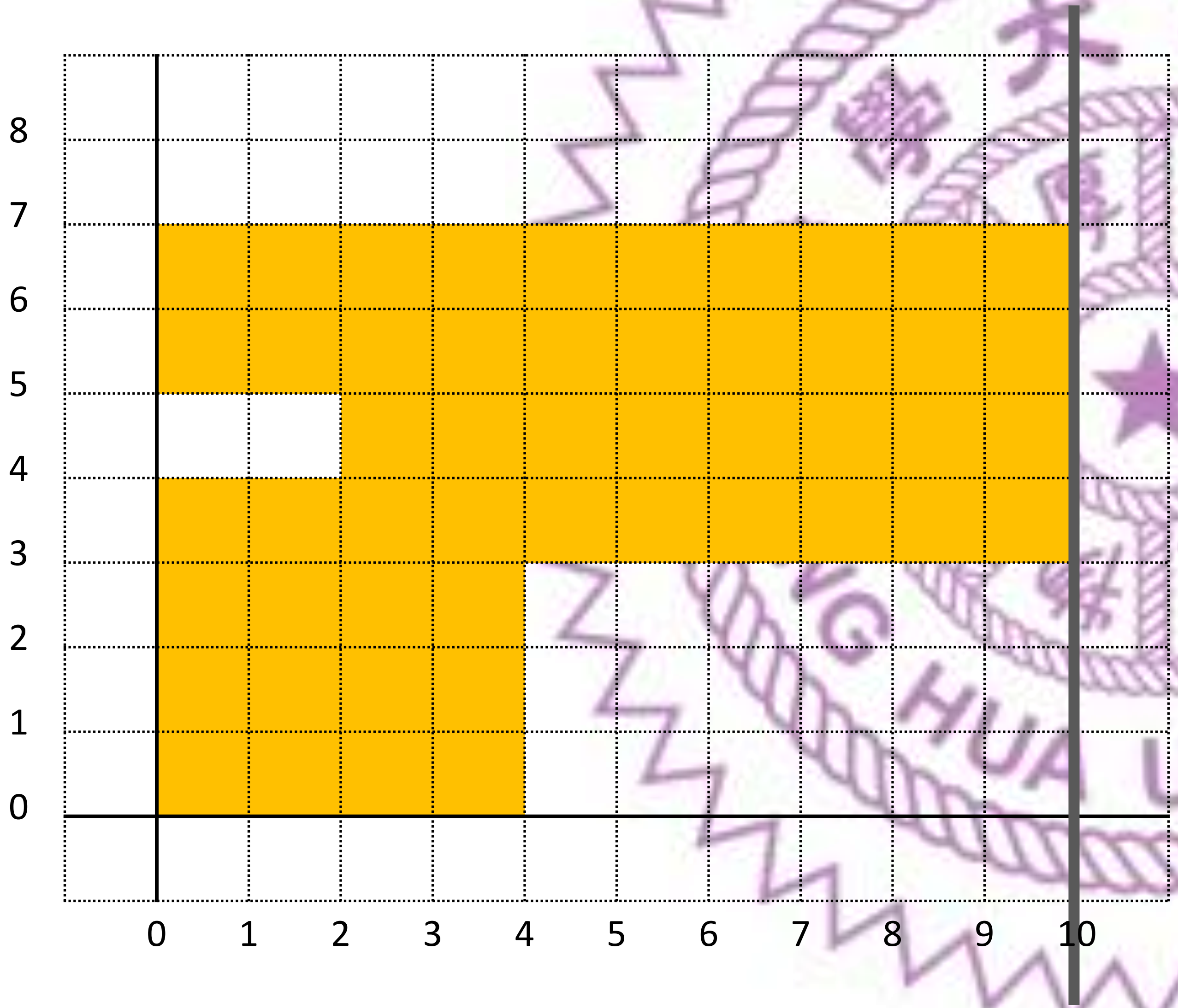
$ans = 50$
$previous\_x = 10$

$ans \mathrel{+}= (10 - 10) \times 2$



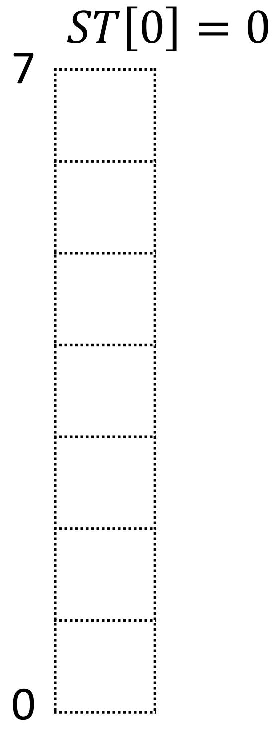

```cpp
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```

想法：用一條線從左掃到右



$ans = 50$
$previous\_x = 10$

$ans \mathrel{+}= (10 - 10) \times 2$
$update(5,7,-1)$
$previous\_x = 10$




```
init(MaxY);
long long previous_x = 0, ans = 0;
for (auto &Seg : Segs) {
  ans += (Seg.x - previous_x) * ST[1].sum;
  update(Seg.y1, Seg.y2, Seg.val);
  previous_x = Seg.x;
}
cout << ans << '\n';
```