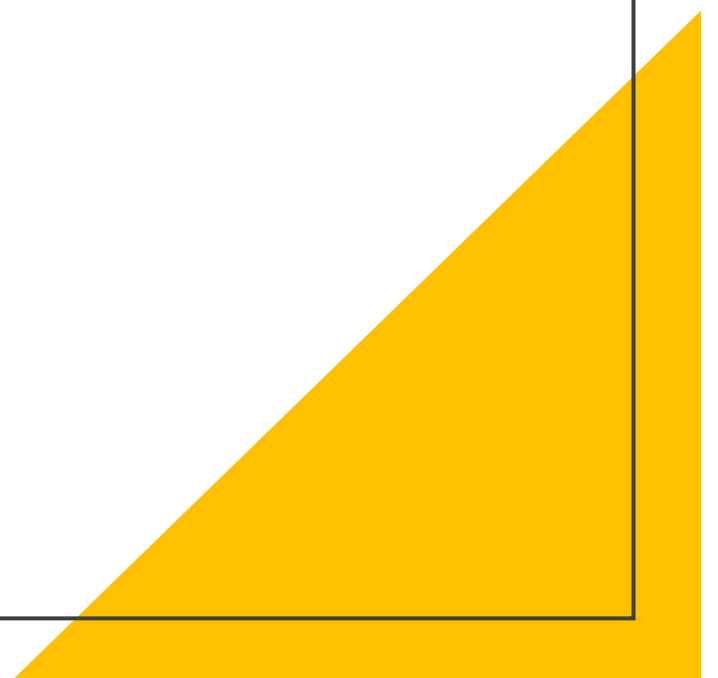


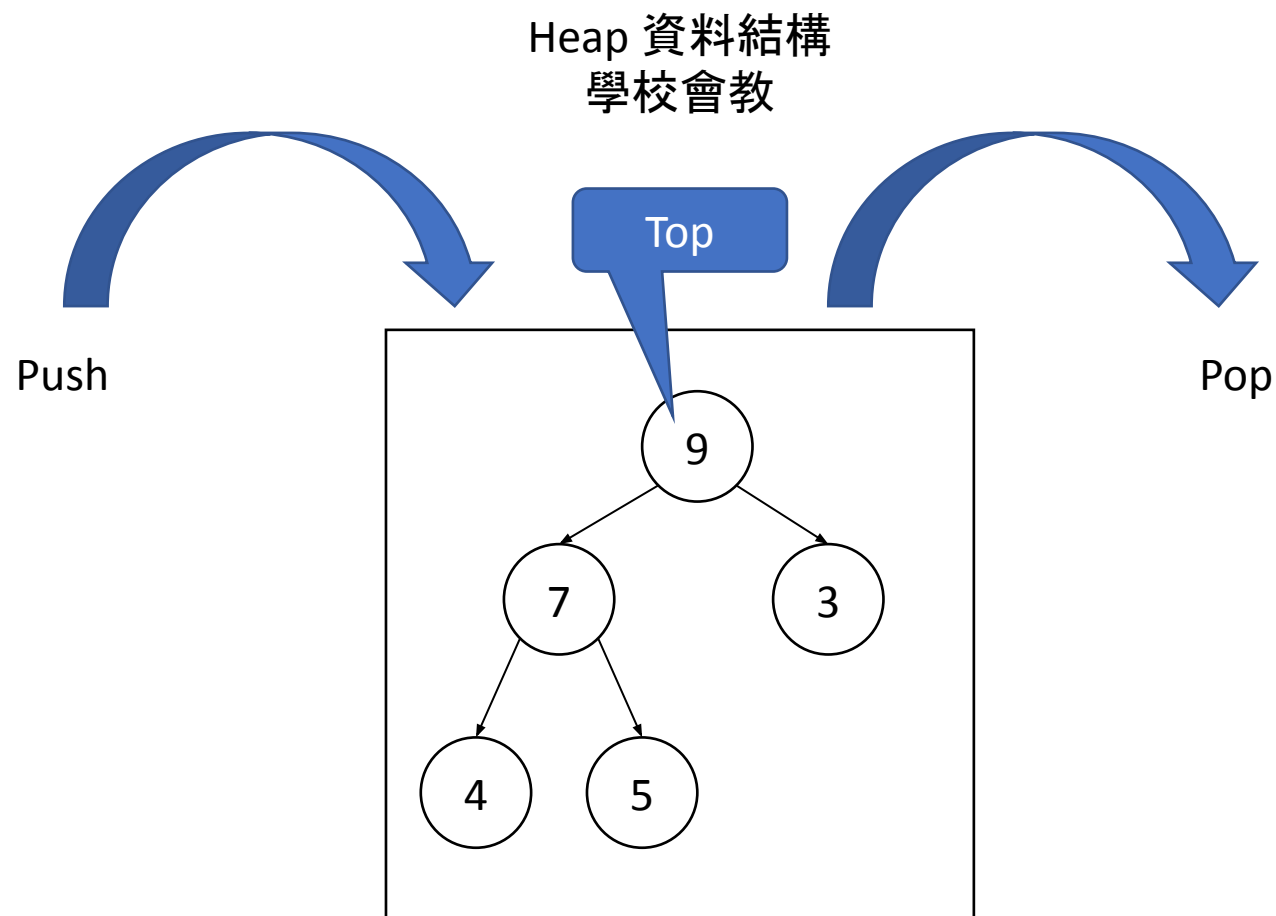
# 基礎資料結構 應用

日月卦長



# std::priority\_queue

- push/ emplace(X) 加入資料
- top() 看最**大**的資料
- pop() 刪掉最**大**的資料



## 元素存取

最大的元素	top()	

## 插入刪除

插入元素	push(x)/emplace(x... )	
刪除最大的元素	pop()	

## 數量資訊

裡面是不是空的	empty()	
裡面有多少東西	size()	

# Priority Queue 使用範例

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
    priority_queue<int> PQ;
    PQ.emplace(1);
    PQ.emplace(3);
    PQ.emplace(2);
    while (!PQ.empty()) {
        cout << PQ.top() << '\n'; // 3 2 1
        PQ.pop();
    }
    return 0;
}
```

# Example 自定義 priority\_queue

- STL priority queue 是最大堆(Max heap)

```
struct CMP {  
    bool operator()(int a, int b) { return a > b; }  
};  
int main() {  
    priority_queue<int> PQ1;  
    priority_queue<int, vector<int>, CMP> PQ2;  
    for (int i = 0; i < 5; ++i) {  
        PQ1.emplace(i);  
        PQ2.emplace(i);  
    }  
    cout << PQ1.top() << ' ' << PQ2.top() << '\n';  
    return 0;  
}
```

帶有operator()的物件稱為函數物件，是一個物件，但使用行為如同函數。



# 常見應用

---

Heap Sort (競賽上不會用到)

---

多路合併

---

管理走訪順序 (延伸: 最短路徑)

---

中位數類題

---

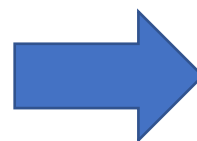
其他

---

# 多路合併 Merge k Sorted Lists

- <https://leetcode.com/problems/merge-k-sorted-lists/>
- 給你 k 個由小到大排序好的 linklist, 請你將這些 linklist 合併成一個小到大排序好的 linklist 後輸出
- Example:
- lists = [[1,4,5],[1,3,4],[2,6]]
  - ans = [1,1,2,3,4,4,5,6]

1->4->5  
1->3->4  
2->6



1->1->2->3->4->4->5->6

# 輸入輸出格式

```
struct ListNode {  
    int val;  
    ListNode *next;  
    ListNode() : val(0), next(nullptr) {}  
    ListNode(int x) : val(x), next(nullptr) {}  
    ListNode(int x, ListNode *next) : val(x), next(next) {}  
};  
  
ListNode* mergeKLists(vector<ListNode*>& lists);
```



# 模擬操作

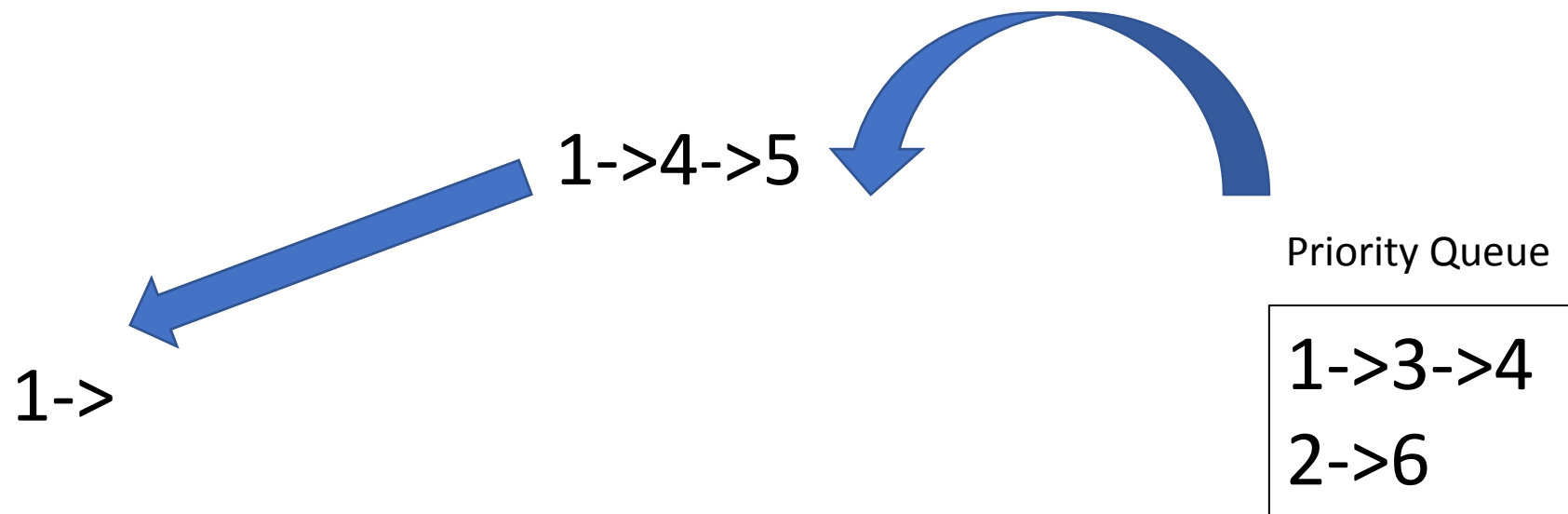
Priority Queue

1->4->5

1->3->4

2->6

# 模擬操作



# 模擬操作

1->

4->5



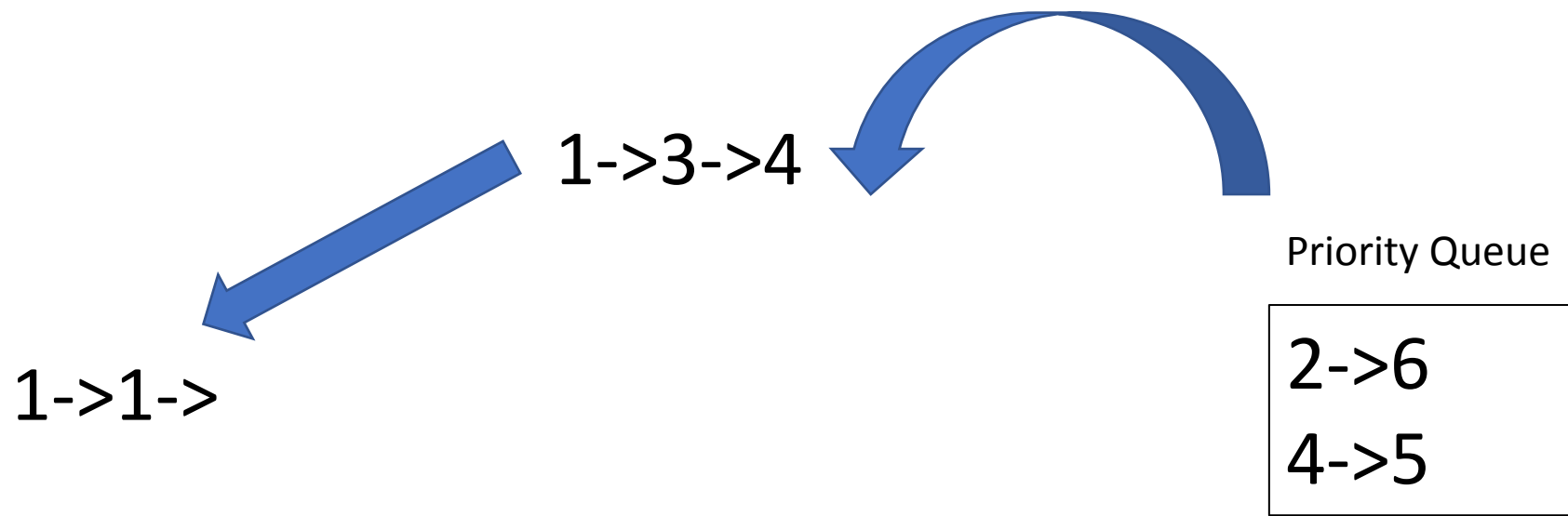
Priority Queue

1->3->4

2->6

4->5

# 模擬操作



# 模擬操作

1->1->

3->4



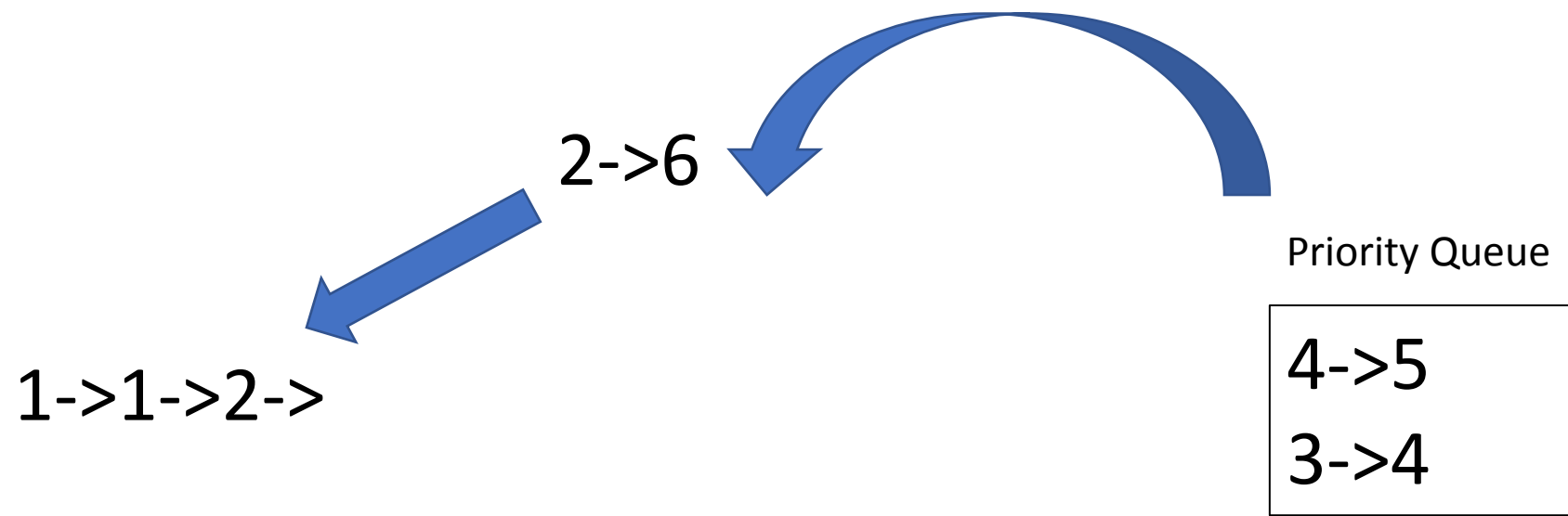
Priority Queue

2->6

4->5

3->4

# 模擬操作



# 模擬操作

1->1->2->



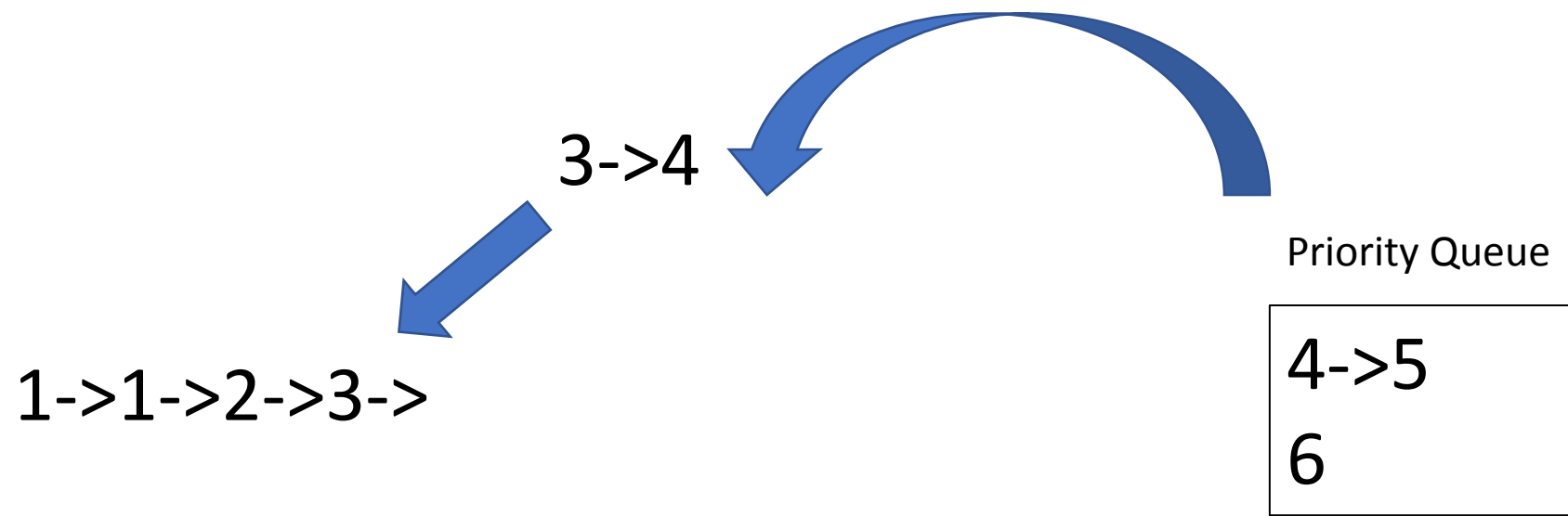
Priority Queue

4->5

3->4

6

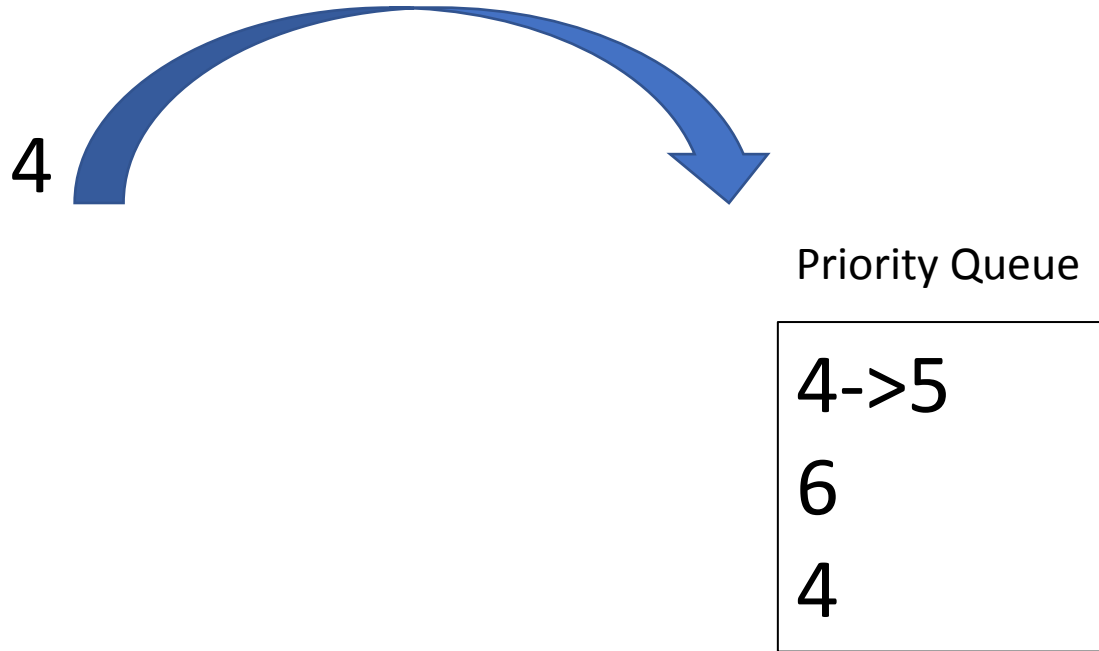
# 模擬操作





# 模擬操作

1->1->2->3->



# 初始化 priority queue

```
struct CMP {  
    bool operator()(ListNode *a, ListNode *b) const { return a->val > b->val; }  
};  
priority_queue<ListNode *, vector<ListNode *>, CMP> PQ;  
void initPQ(vector<ListNode *> &lists) {  
    decltype(PQ)().swap(PQ);  
    for (auto nd : lists)  
        if (nd) PQ.emplace(nd);  
}
```

# 關鍵程式碼

```
ListNode *mergeKLists(vector<ListNode *> &lists) {  
    initPQ(lists);  
    ListNode *root = nullptr;  
    ListNode **cur = &root;  
    while (PQ.size()) {  
        auto nd = PQ.top();  
        PQ.pop();  
        *cur = nd;  
        cur = &nd->next;  
        if (*cur) PQ.emplace(*cur);  
    }  
    return root;  
}
```

# 不使用額外空間 (面試題)

```
ListNode *mergeKLists(vector<ListNode *> &lists) {  
    auto iter = remove_if(lists.begin(), lists.end(), [](auto x) { return !x; });  
    lists.erase(iter, lists.end());  
    make_heap(lists.begin(), lists.end(), CMP());  
    ListNode *root = nullptr;  
    ListNode **cur = &root;  
    while (lists.size()) {  
        auto nd = lists.front();  
        pop_heap(lists.begin(), lists.end(), CMP());  
        lists.pop_back();  
        *cur = nd;  
        cur = &nd->next;  
        if (*cur) {  
            lists.emplace_back(*cur);  
            push_heap(lists.begin(), lists.end(), CMP());  
        }  
    }  
    return root;  
}
```

# 管理走訪順序 (延伸: 最短路徑)

- <https://leetcode.com/problems/find-the-kth-smallest-sum-of-a-matrix-with-sorted-rows/>
- 給你一個每個 row 都由小到大排的矩陣 mat, 問你從每個 row 選一個數字, 總和第 k 小的是多少
- Example:
  - mat = [[1,3,11],[2,4,6]], k = 5
    - ans = 7
  - mat = [[1,3,11],[2,4,6]], k = 9
    - ans = 17

mat	0	1	2
0	1	3	11
1	2	4	6

前五小的組合: [1,2], [1,4], [3,2], [3,4], [1,6]

# 觀察

- $\text{mat} = [[1,10,10],[1,4,5],[2,3,6]]$
- 觀察前幾小的狀態

mat	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6

$K = 1$

• [1,1,2]

mat	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6

$K = 2$

• [1,1,3]

mat	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6



$K = 3$

• [1,4,2]

mat	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6

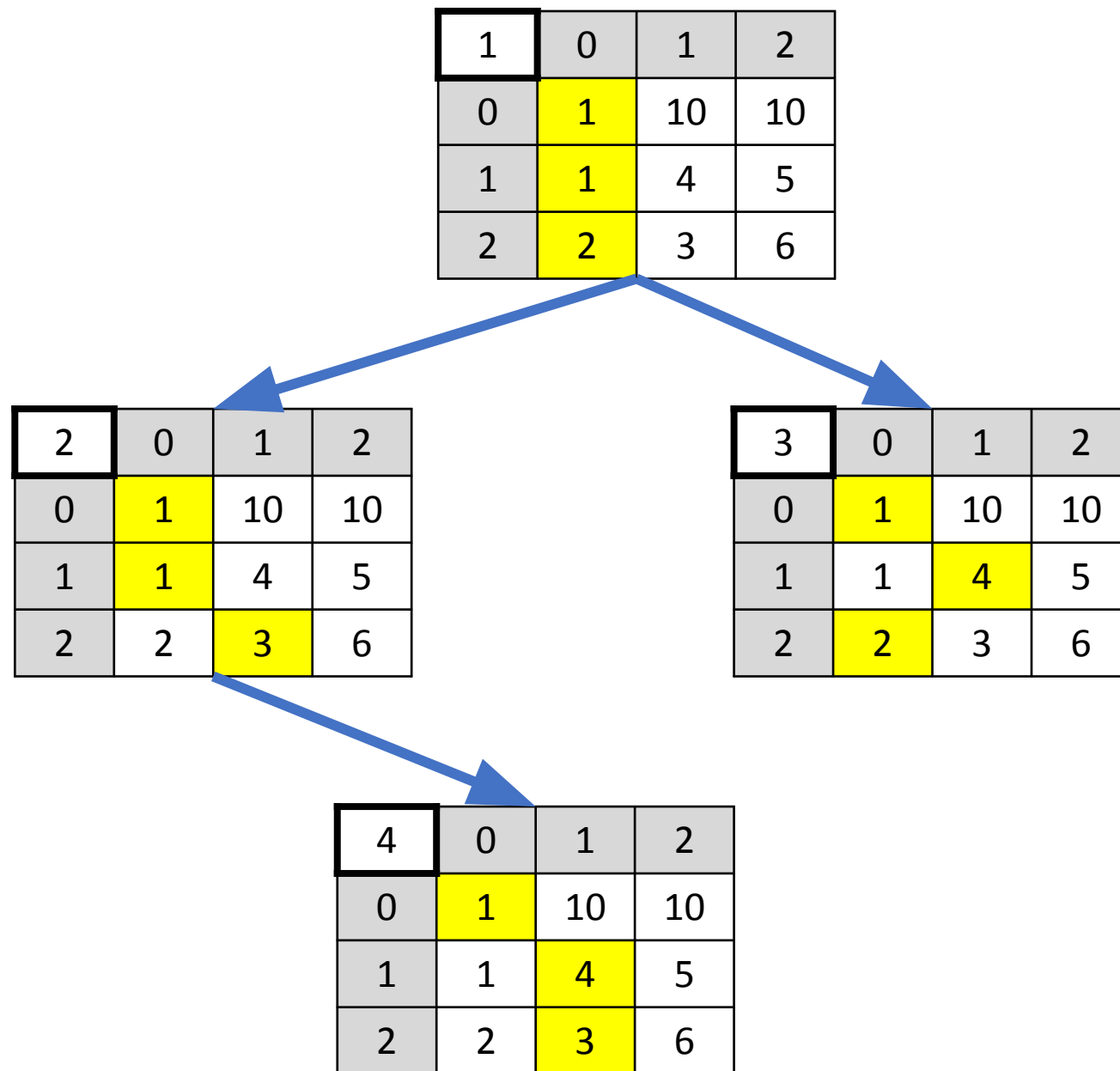
$K = 4$

• [1,4,3]

mat	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6

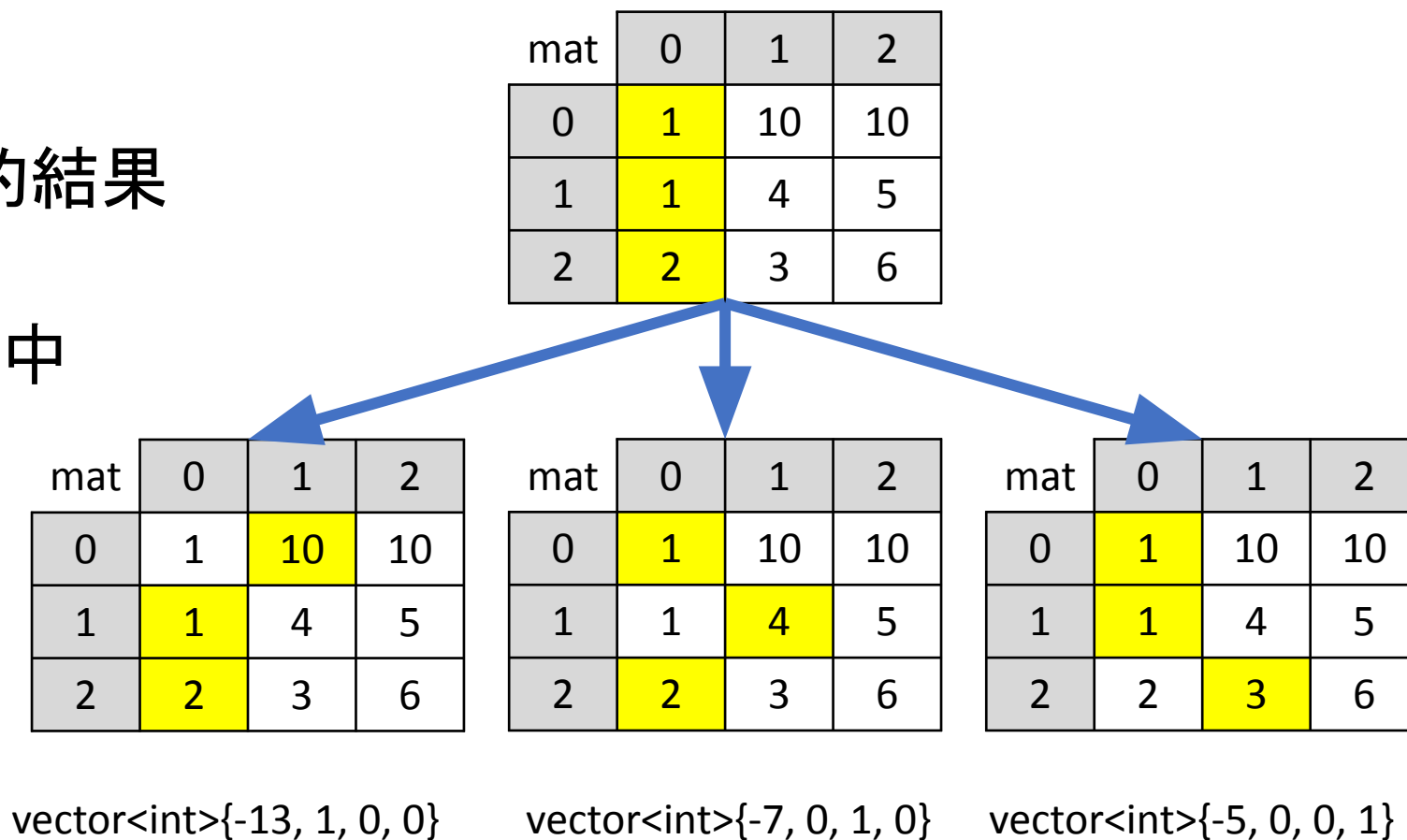
# 觀察

- 第  $k+1$  小的答案
- 是前  $k$  小的答案中
- 把某 row 的  $\text{index}+1$  後得到



# 作法

- 用 priority queue 存資料
- 在知道第 k 小的答案後  
將每個 row 的 index+1 的結果  
存到 priority queue 中
- 下一輪從 priority queue 中  
取出第 k+1 小的答案
- 一直做下去



# 不同狀態有機會產生相同的結果

- 用 `set<vector<int>>` 判斷有沒有重複計算

`vector<int>{-5, 0, 1, 0}`

2	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6

`vector<int>{-7, 0, 1, 0}`

3	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6



4	0	1	2
0	1	10	10
1	1	4	5
2	2	3	6

`vector<int>{-8, 0, 1, 0}`

# 程式碼

```
int kthSmallest(vector<vector<int>> &mat, int k) {
    int m = mat.size(), n = mat[0].size();
    priority_queue<vector<int>> PQ;
    vector<int> cur(m + 1, 0);
    for (auto &v: mat) cur[0] -= v[0];
    PQ.emplace(cur);
    set<vector<int>> visited{{cur}};
    for(int t = 1; t < k; ++t) {
        cur = PQ.top(), PQ.pop();
        for (int i = 1; i <= m; ++i) {
            if (cur[i] == n - 1) continue;
            auto next = cur;
            ++next[i];
            next[0] -= (mat[i - 1][next[i]] - mat[i - 1][next[i] - 1]);
            if (!visited.emplace(next).second) continue;
            PQ.emplace(move(next));
        }
    }
    return -PQ.top().at(0);
}
```

# 另一種二分搜的做法 (非本單元主題)

```
int n, m;
vector<vector<int>> *A;
int total, K;
bool dfs(int i, int sum, int limit) {
    if (i >= n) {
        total += sum <= limit;
        return sum <= limit;
    }
    bool ans = false;
    for (auto x : A->at(i)) {
        auto cur = dfs(i + 1, sum + x,
limit);
        ans |= cur;
        if (!cur || total > K) break;
    }
    return ans;
}
```

```
int kthSmallest(
    vector<vector<int>> &mat, int k) {
    n = mat.size(), m = mat[0].size();
    int L = 0, R = 0;
    for (auto &V : mat) R += V.back();
    K = k;
    A = &mat;
    tie(ignore, R) =
    binarySearch(L, R, [&](int mid) {
        total = 0;
        dfs(0, 0, mid);
        return total < k;
    });
    return R;
}
```

# 中位數 (Zerojudge D713)

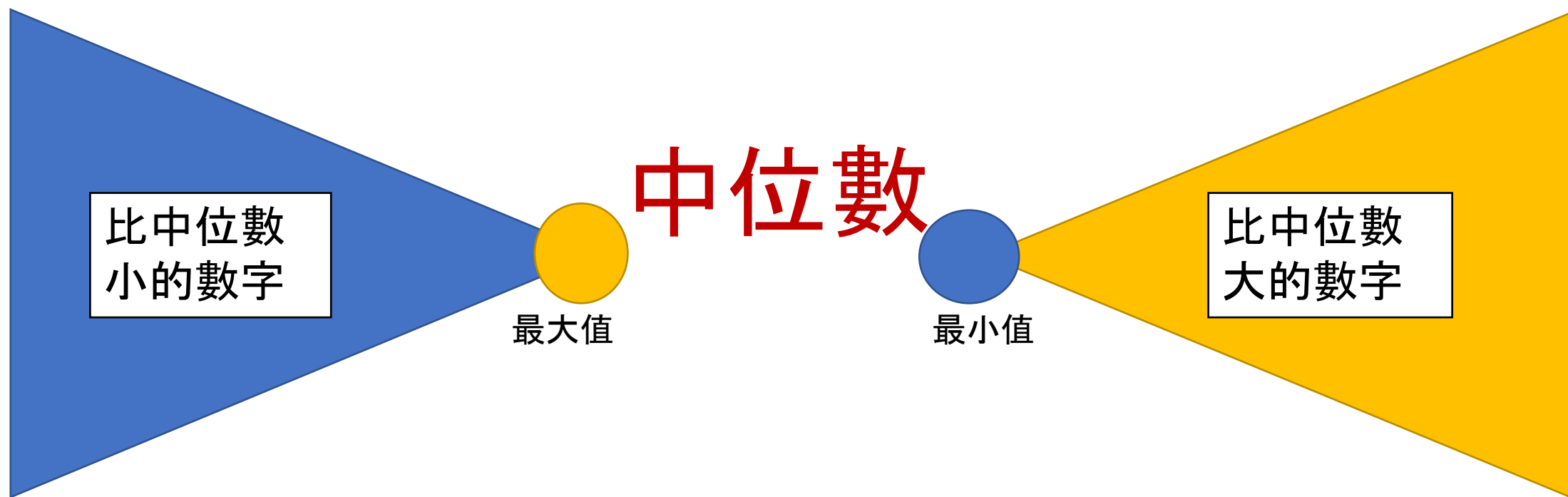
中位數

Zerojudge D713

依序讀取一個數列  $a_1, a_2, a_3, a_4, a_5 \cdots a_n$ ，每讀取一個數字，請回答到目前為止的中位數是多少。若有答案是小數請直接忽略到整數位。 $(n < 200000)$

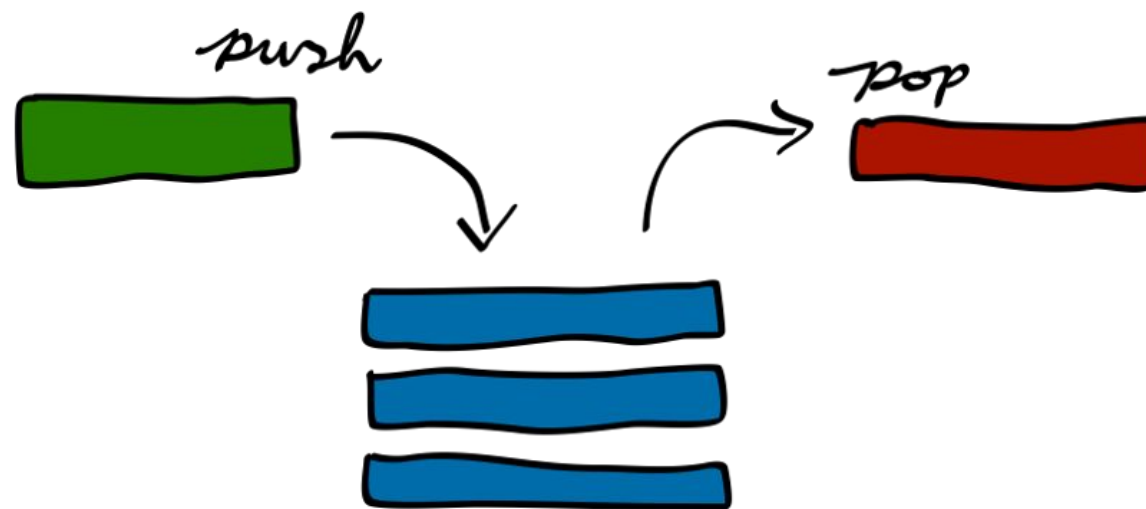


# 中位數

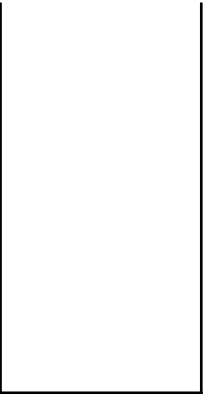


# Stack

- push/emplace(X) 加入資料X
- top() 看最上面的資料
- pop() 刪掉最上面的資料
- First in last out



# std::stack



```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> STK;
    STK.emplace(123);
    STK.emplace(7122);
    STK.pop();
    STK.emplace(456);
    cout << STK.top() << '\n';
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> STK;
    STK.emplace_back(123);
    STK.emplace_back(7122);
    STK.pop_back();
    STK.emplace_back(456);
    cout << STK.back() << '\n';
    return 0;
}
```

# std::stack

123

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> STK;
    STK.emplace(123);
    STK.emplace(7122);
    STK.pop();
    STK.emplace(456);
    cout << STK.top() << '\n';
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> STK;
    STK.emplace_back(123);
    STK.emplace_back(7122);
    STK.pop_back();
    STK.emplace_back(456);
    cout << STK.back() << '\n';
    return 0;
}
```

# std::stack

7122  
123

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> STK;
    STK.emplace(123);
    STK.emplace(7122);
    STK.pop();
    STK.emplace(456);
    cout << STK.top() << '\n';
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> STK;
    STK.emplace_back(123);
    STK.emplace_back(7122);
    STK.pop_back();
    STK.emplace_back(456);
    cout << STK.back() << '\n';
    return 0;
}
```

# std::stack

123

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> STK;
    STK.emplace(123);
    STK.emplace(7122);
    STK.pop();
    STK.emplace(456);
    cout << STK.top() << '\n';
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> STK;
    STK.emplace_back(123);
    STK.emplace_back(7122);
    STK.pop_back();
    STK.emplace_back(456);
    cout << STK.back() << '\n';
    return 0;
}
```

# std::stack

456

123

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> STK;
    STK.emplace(123);
    STK.emplace(7122);
    STK.pop();
    STK.emplace(456);
    cout << STK.top() << '\n';
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> STK;
    STK.emplace_back(123);
    STK.emplace_back(7122);
    STK.pop_back();
    STK.emplace_back(456);
    cout << STK.back() << '\n';
    return 0;
}
```

# 其實是用 deque 做出來的

```
1  #include <iostream>
2  #include <vector>
3  #include <stack> <typename _Tp, typename _Sequence = deque < _Tp >>
4  using namespace std;
5
6  int main() {
7      stack<int,>
8  }
```

A standard container giving FILO behavior.



# 讓 std::stack 內部用 vector

```
#include <iostream>
#include <stack>
#include <vector>
using namespace std;
int main() {
    stack<int, vector<int>> STK;
    STK.emplace(123);
    STK.emplace(7122);
    STK.pop();
    STK.emplace(456);
    cout << STK.top() << '\n';
    return 0;
}
```

## 元素存取

	stack
頭部	top()

## 數量資訊

	stack
裡面是不是空的	empty()
裡面有多少東西	size()

# 括號匹配問題(講義)

## 例題演練

Parentheses Balance

UVa 673

今天姆咪正在寫程式，不過由於姆咪是 \*\* 很粗心的人 \*\*，把文件中所有括號的配對都亂掉了，已知文件中的括號只有小括號 `()` 以及中括號 `[]`，給出文件的括號序列，能檢查括號是否有正確配對嗎？

定義一個合法的括號配對字串如下

1. 空字串是合法的括號字串
2. 如果  $A, B$  是合法的括號字串，那  $AB$  也是合法的括號字串
3. 如果  $A$  是合法的括號字串，那  $(A)$ 、 $[A]$  也是合法的括號字串

example for  $([])[ ]$



$([])[ ]$

example for  $([])[]$

(



$([])[]$

example for  $([])[]$

( [
-----



$([])[]$

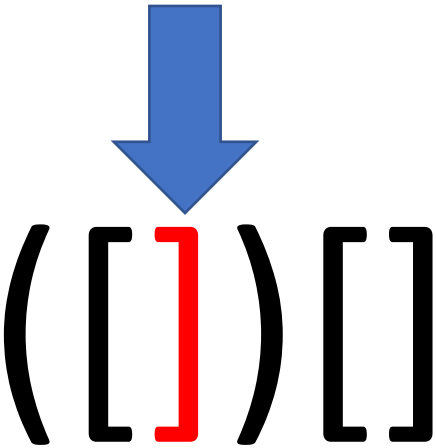
example for  $([])[]$

( [

( [ ] ) []

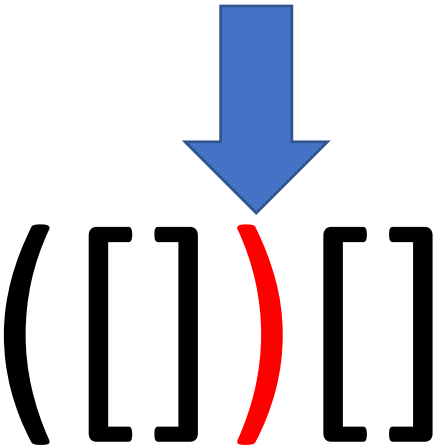
example for  $([])[]$

(

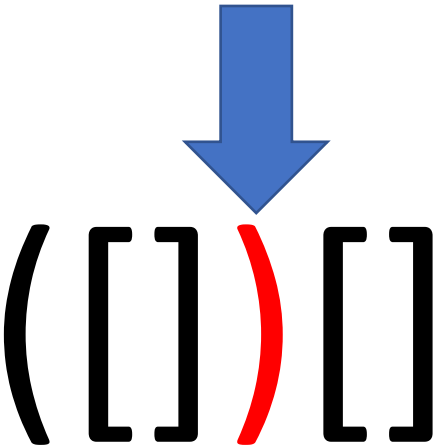




example for ([ ])[ ]




example for  $([])[]$

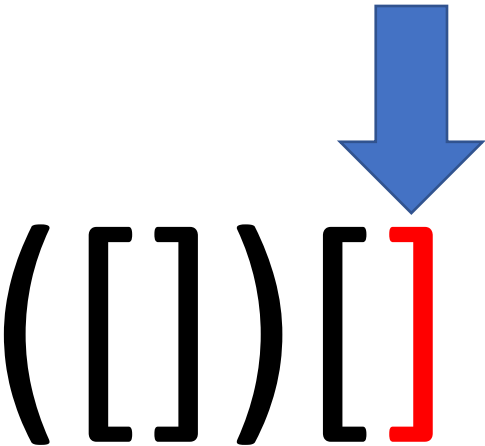


example for  $([])[]$

[

  
 $([])[]$

example for ([ ])[ ]



example for ([ ])[ ]



([ ])[ ]



# Monotonic Stack

保證 stack 中的  
值遞增或遞減

去除永遠不可能  
成為答案的元素



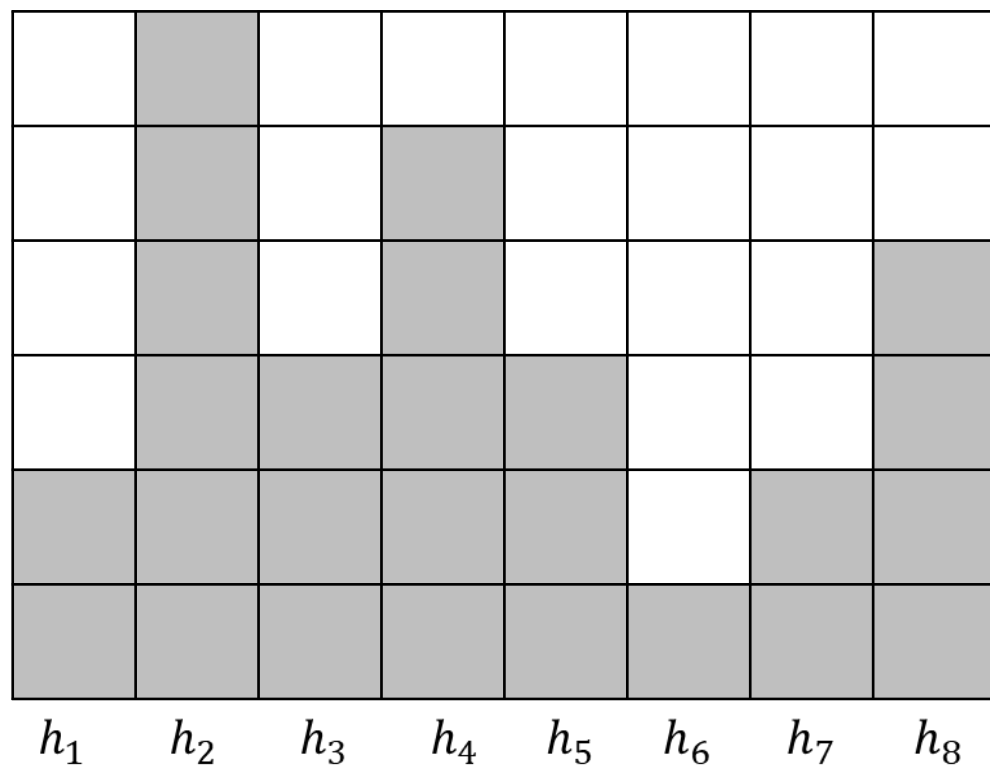
# 乳牛(POJ 3250)

Bad Hair Day

POJ 3250

有  $n$  ( $n \leq 80000$ ) 隻乳牛站在一條直線上，乳牛的身高由左而右依序為  $h_1, h_2, \dots, h_n$ ，請為每一隻乳牛向左看可以看到幾隻乳牛？  
一隻乳牛可以看到左邊所有的乳牛，至到有一隻乳牛的身高大於等於該乳牛。

考慮以下乳牛



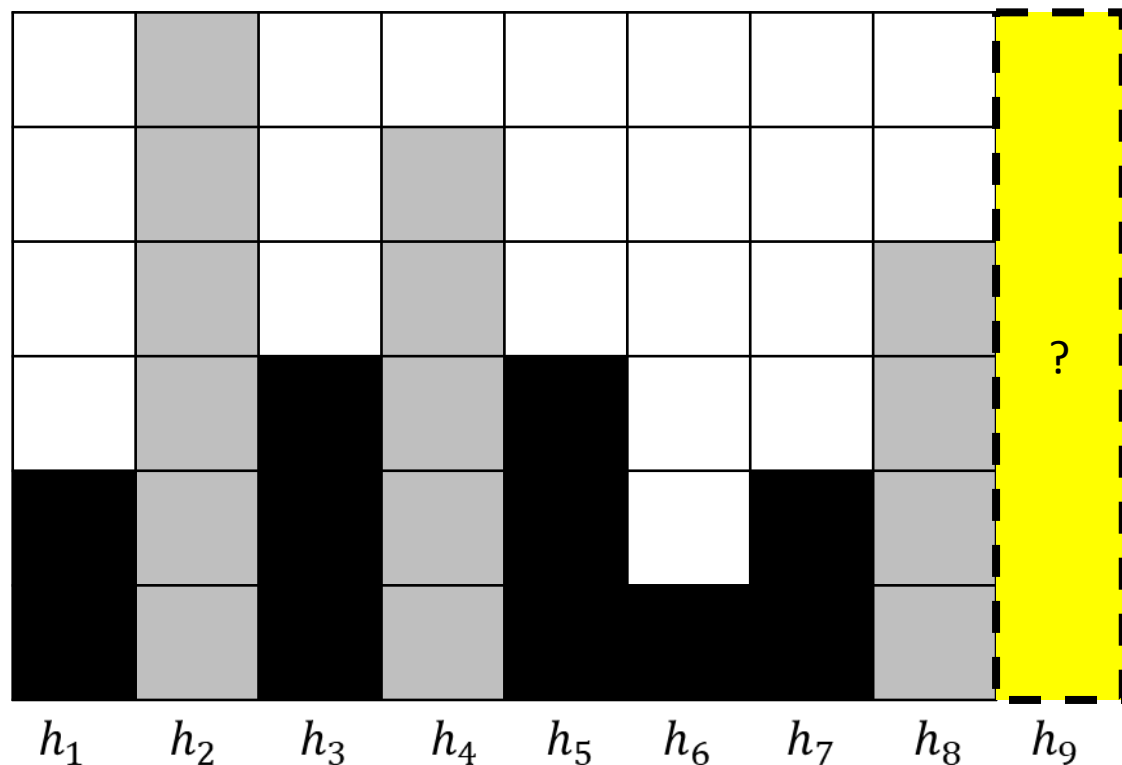




# 假設有個不知道高度的 $h_9$

能不能將絕對不可能是  $h_9$  能看到最遠的乳牛刪除呢？

無論  $h_9$  高度是多少  
 $h_1, h_3, h_5, h_6, h_7$  絕對不可能  
是  $h_9$  能看到最遠的乳牛

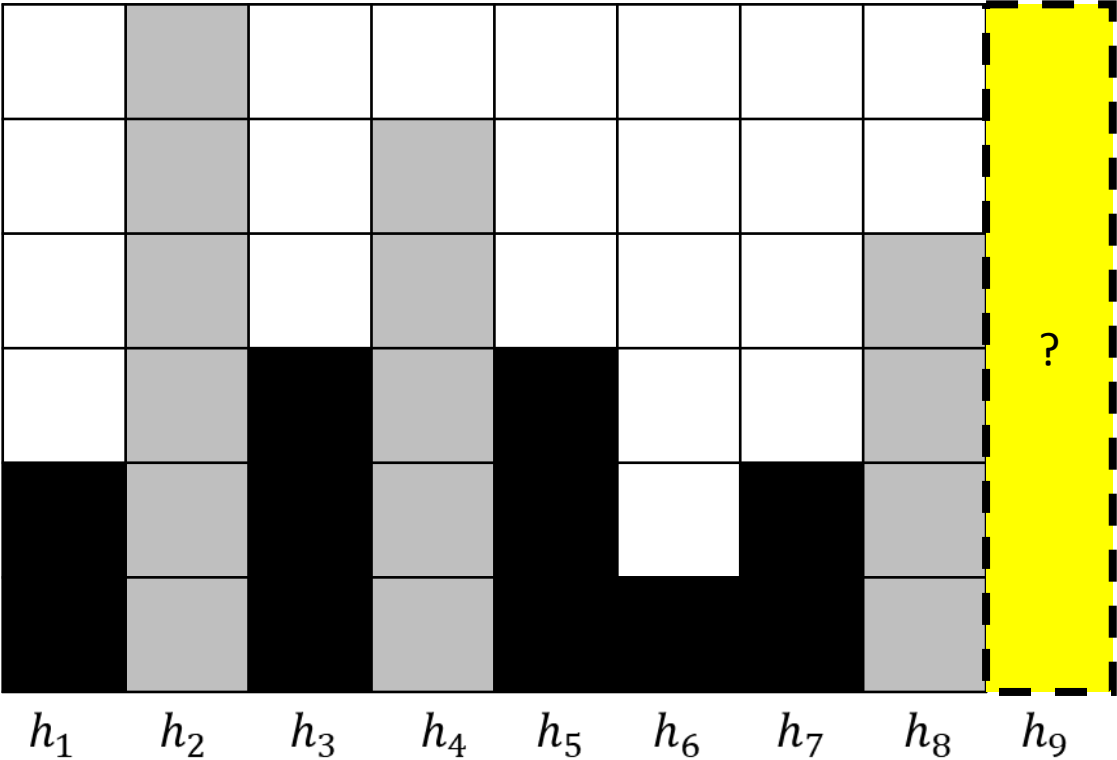


剩下的  $h_2, h_4, h_8$   
會呈現非嚴格遞增的關係

# 計算 $h_9$ 往左看能看到幾隻牛

STK

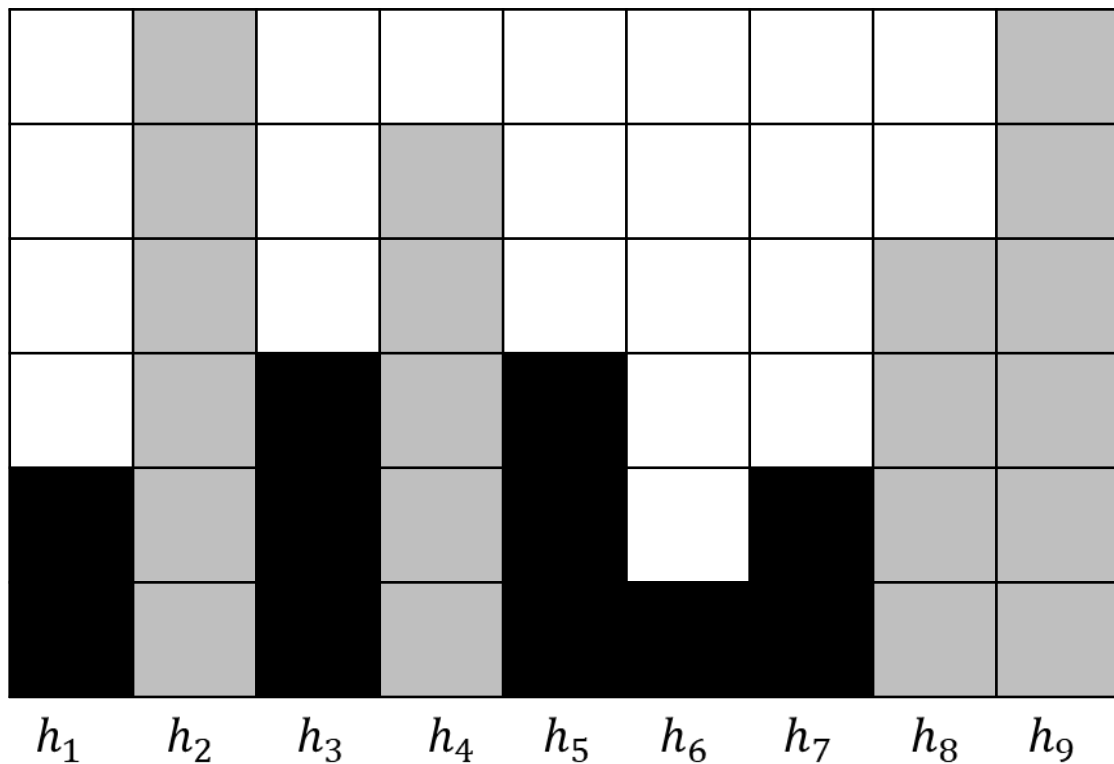
用 stack 維護有機會被下隻牛看到的牛



# 計算 $h_9$ 往左看能看到幾隻牛

STK

用 stack 維護有機會被下隻牛  
看到的牛

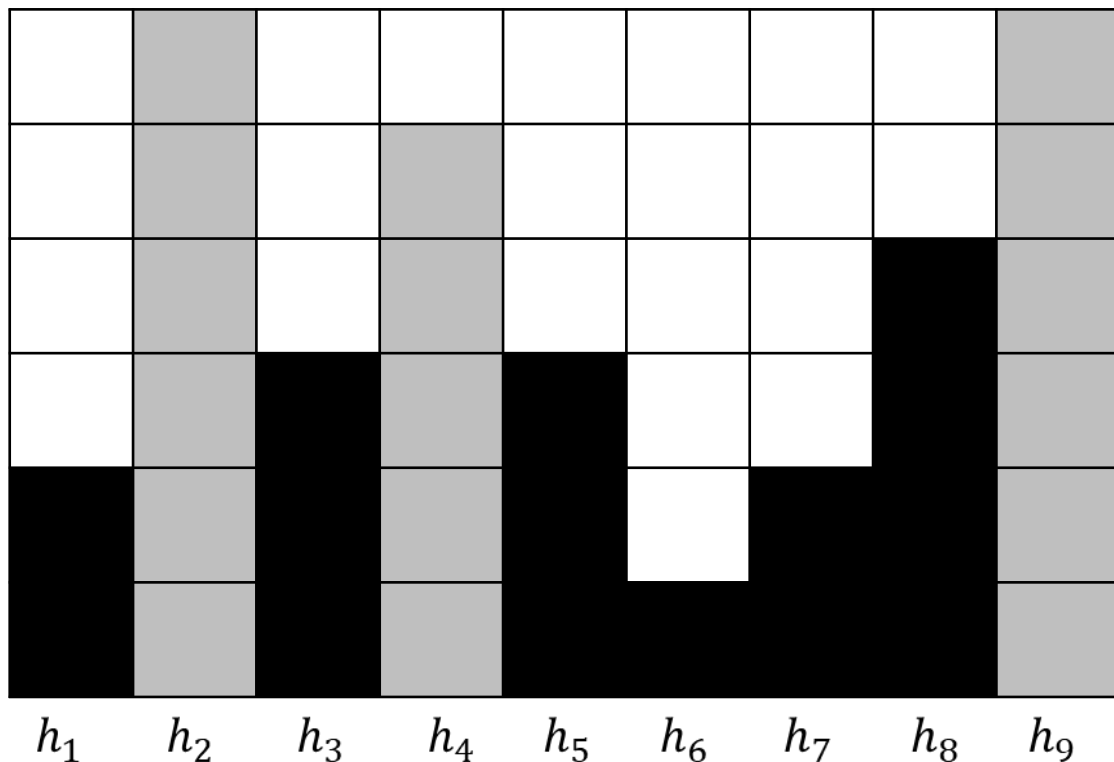


假設  $h_9 = 6$

# 計算 $h_9$ 往左看能看到幾隻牛

STK

用 stack 維護有機會被下隻牛看到的牛

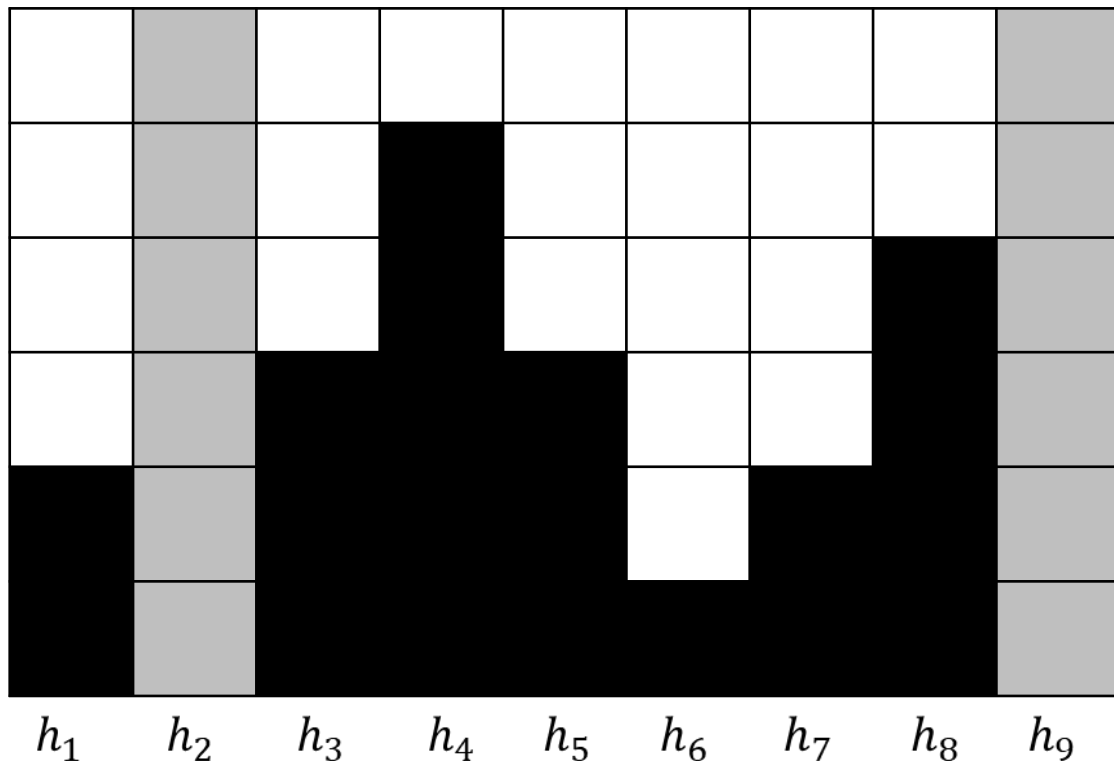


假設  $h_9 = 6$   
STK 頂端是  $h_8$   
 $h_8 < h_9$  所以  $h_8$  沒用了

# 計算 $h_9$ 往左看能看到幾隻牛

STK

用 stack 維護有機會被下隻牛看到的牛



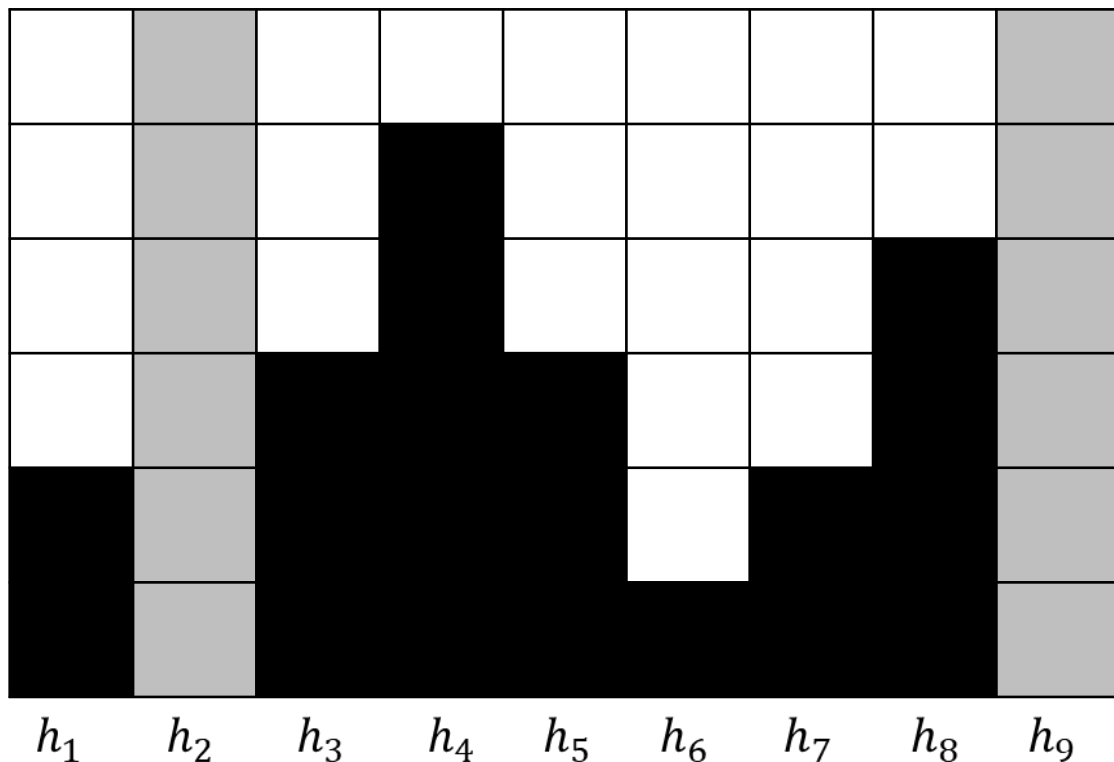
假設  $h_9 = 6$   
STK 頂端是  $h_8$   
 $h_8 < h_9$  所以  $h_8$  沒用了  
STK 頂端是  $h_4$   
 $h_4 < h_9$  所以  $h_4$  沒用了

# 計算 $h_9$ 往左看能看到幾隻牛

STK

9

用 stack 維護有機會被下隻牛看到的牛



假設  $h_9 = 6$

STK 頂端是  $h_8$

$h_8 < h_9$  所以  $h_8$  沒用了

STK 頂端是  $h_4$

$h_4 < h_9$  所以  $h_4$  沒用了

STK 頂端是  $h_2$

$h_2 \neq h_9$  所以  $h_9$  最遠能看到  $h_2$

把  $h_9$  放進 STK 為  $h_{10}$  做準備

# 實際程式碼有三個部分

刪除沒用的東西

計算  $i$  的答案

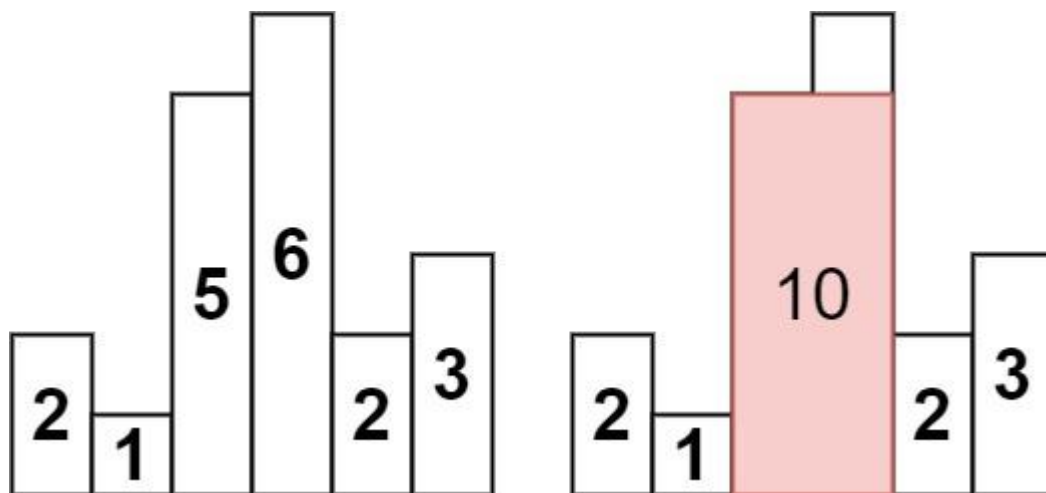
將  $i$  放進 STK

```
void solve(vector<int> h) {  
    stack<int> STK;  
    for (size_t i = 0; i < h.size(); ++i) {  
        while (STK.size() && h[STK.top()] < h[i]) STK.pop();  
  
        if (STK.empty()) cout << i << endl;  
        else cout << i - STK.top() << endl;  
  
        STK.emplace(i);  
    }  
}
```

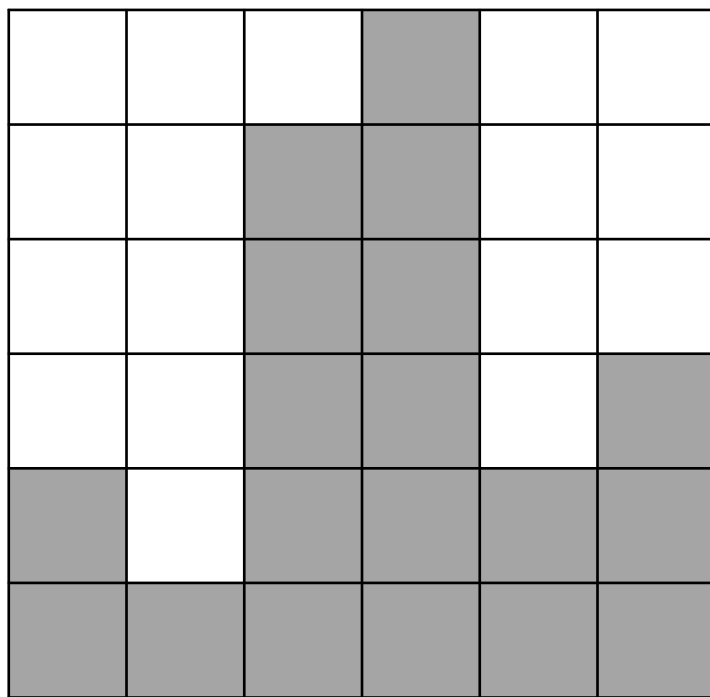


# 最大矩形

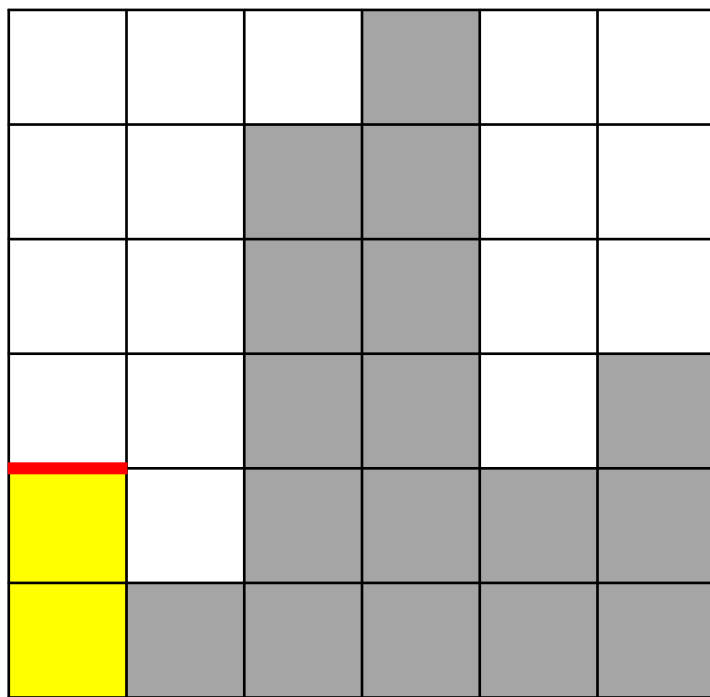
- <https://leetcode.com/problems/largest-rectangle-in-histogram/>
- 告訴你每根長條的高度，問你面積最大的矩形的面積
- heights = [2,1,5,6,2,3]



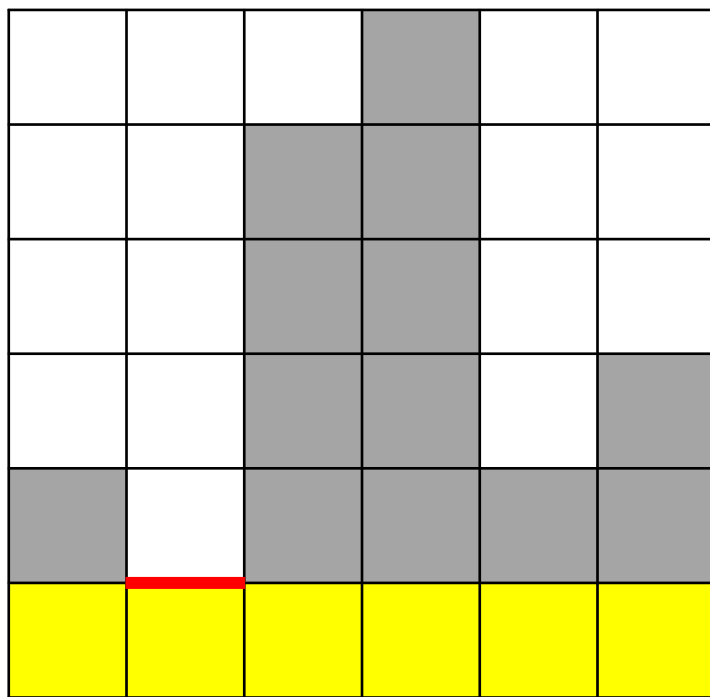
# 觀察所有邊界最大化的矩形



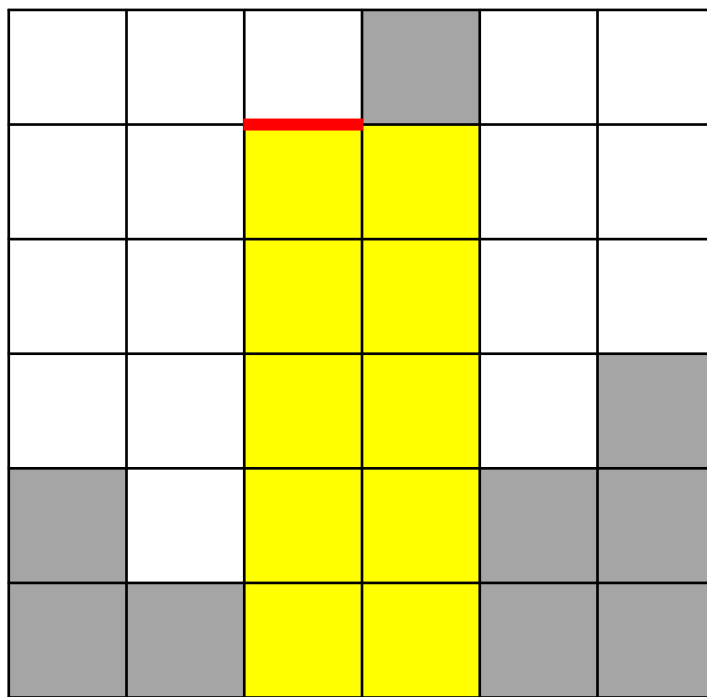
# 觀察所有邊界最大化的矩形



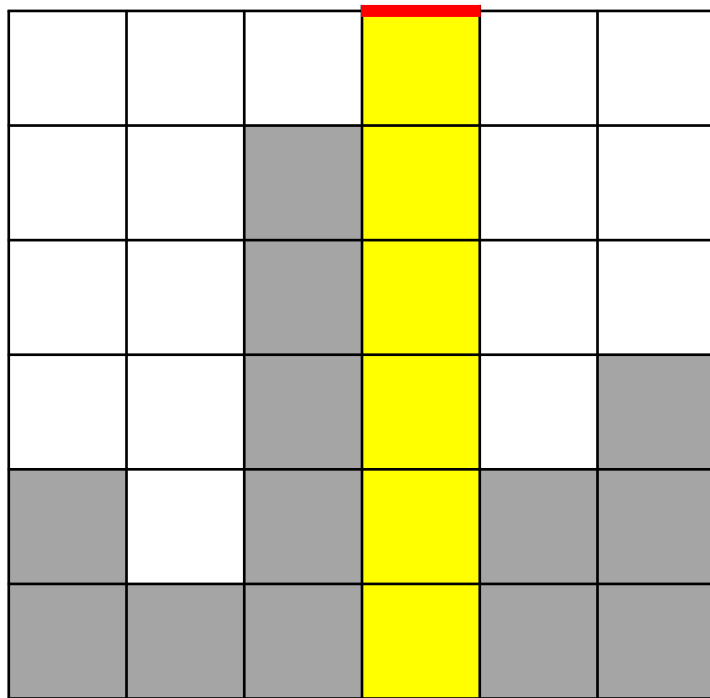
# 觀察所有邊界最大化的矩形



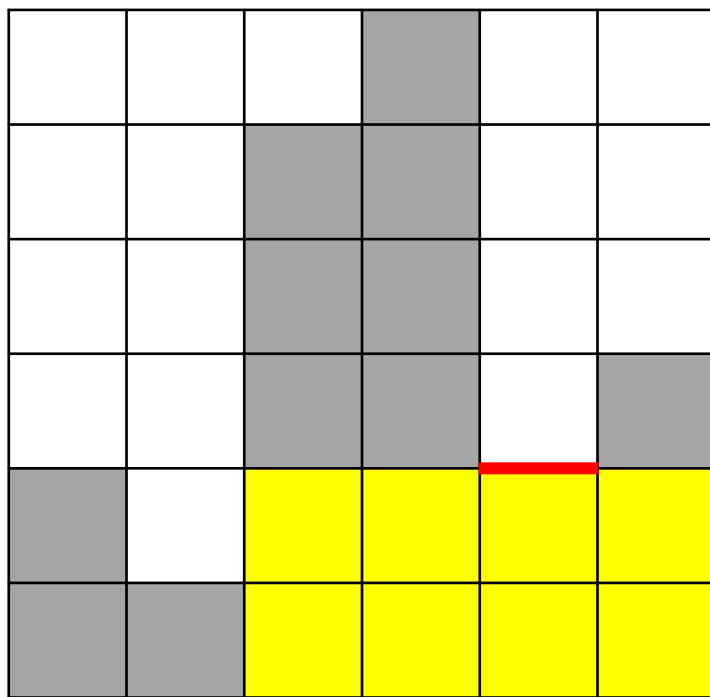
# 觀察所有邊界最大化的矩形



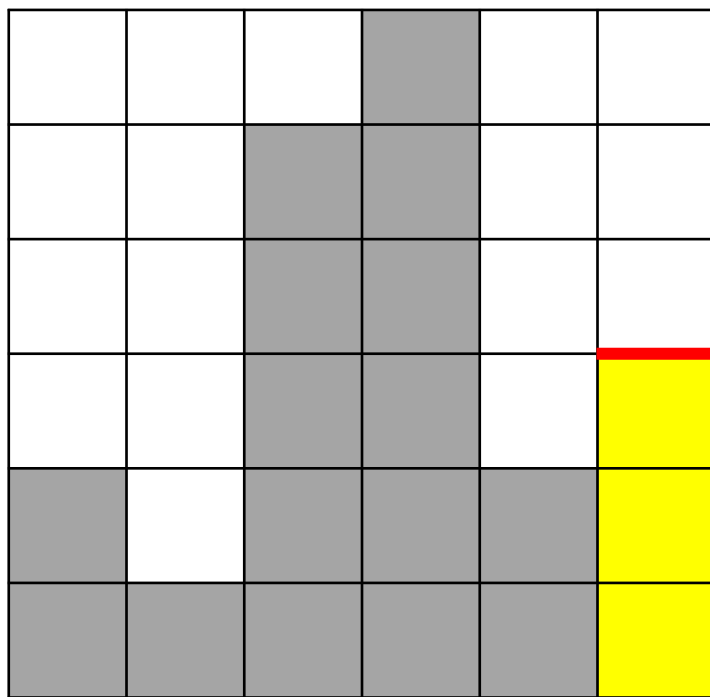
# 觀察所有邊界最大化的矩形



# 觀察所有邊界最大化的矩形



# 觀察所有邊界最大化的矩形





# 性質

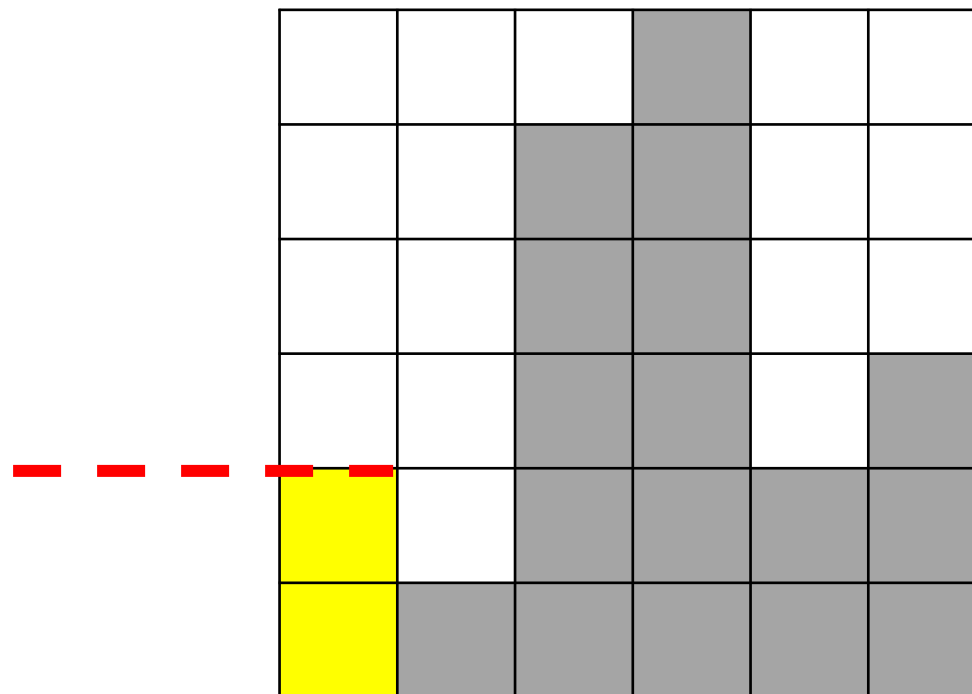
---

- 邊界最大化的矩形最多只有  $n$  個



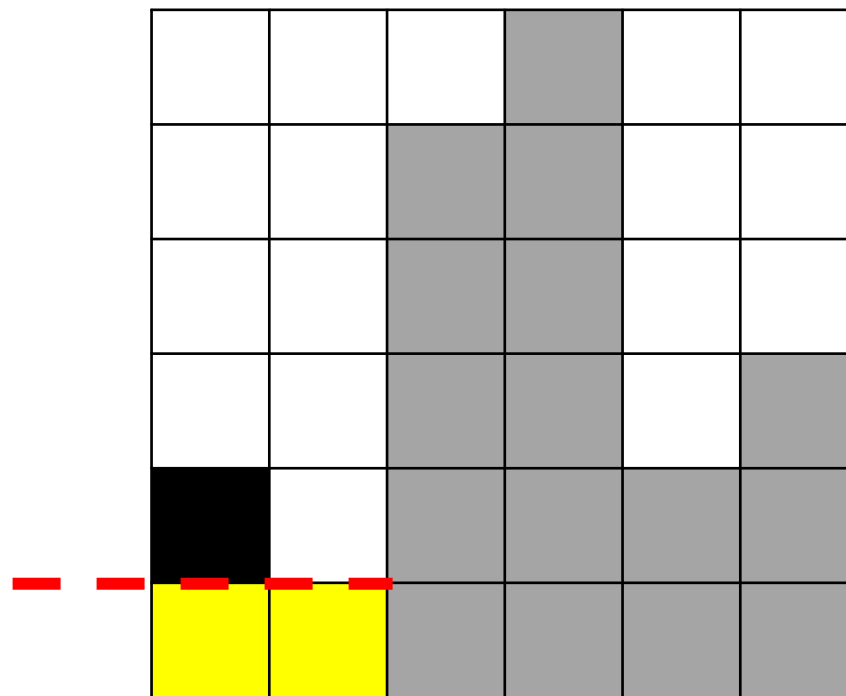
## 去除不可能成為答案的「區域」

- 由左到右依序加入長條
- 每個長條被加入時  
將左邊比他高的部分「砍掉」



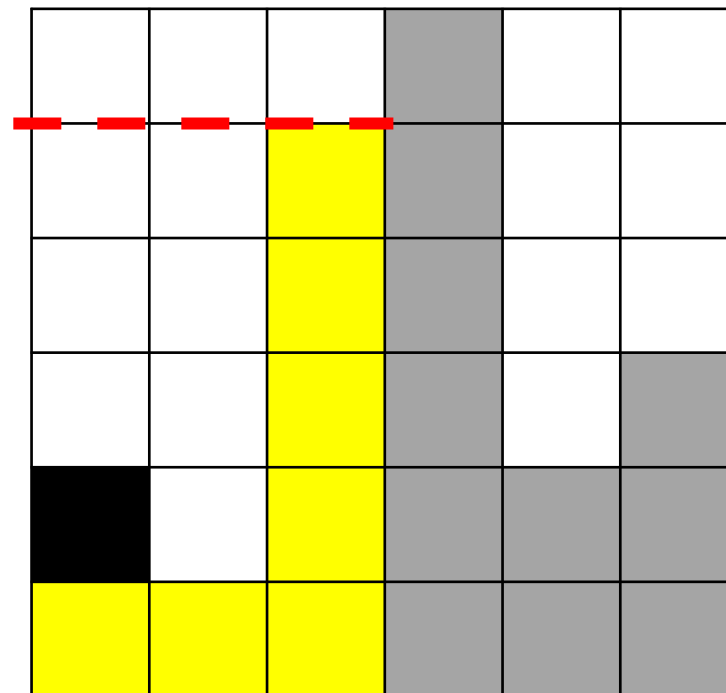
# 去除不可能成為答案的「區域」

- 由左到右依序加入長條
- 每個長條被加入時  
將左邊比他高的部分「砍掉」



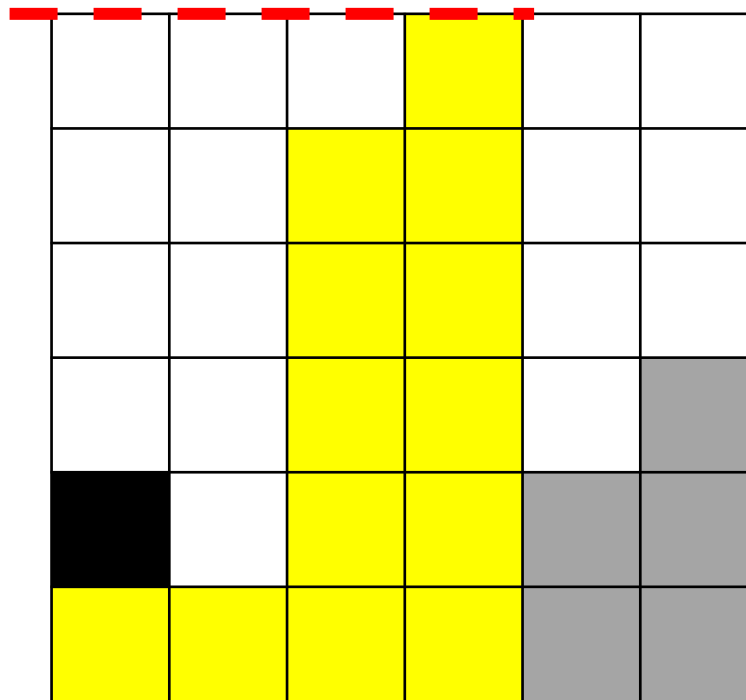
# 去除不可能成為答案的「區域」

- 由左到右依序加入長條
- 每個長條被加入時  
將左邊比他高的部分「砍掉」



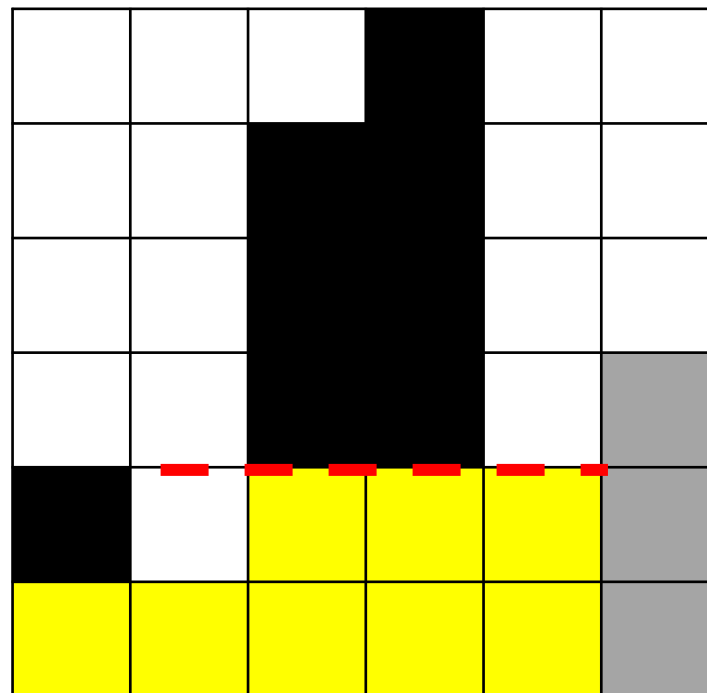
# 去除不可能成為答案的「區域」

- 由左到右依序加入長條
- 每個長條被加入時  
將左邊比他高的部分「砍掉」



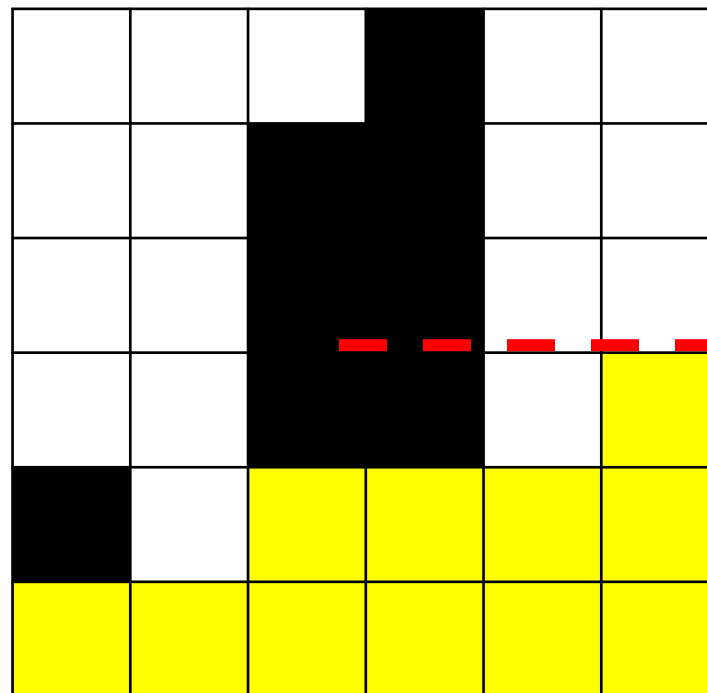
# 去除不可能成為答案的「區域」

- 由左到右依序加入長條
- 每個長條被加入時  
將左邊比他高的部分「砍掉」



# 去除不可能成為答案的「區域」

- 由左到右依序加入長條
- 每個長條被加入時  
將左邊比他高的部分「砍掉」

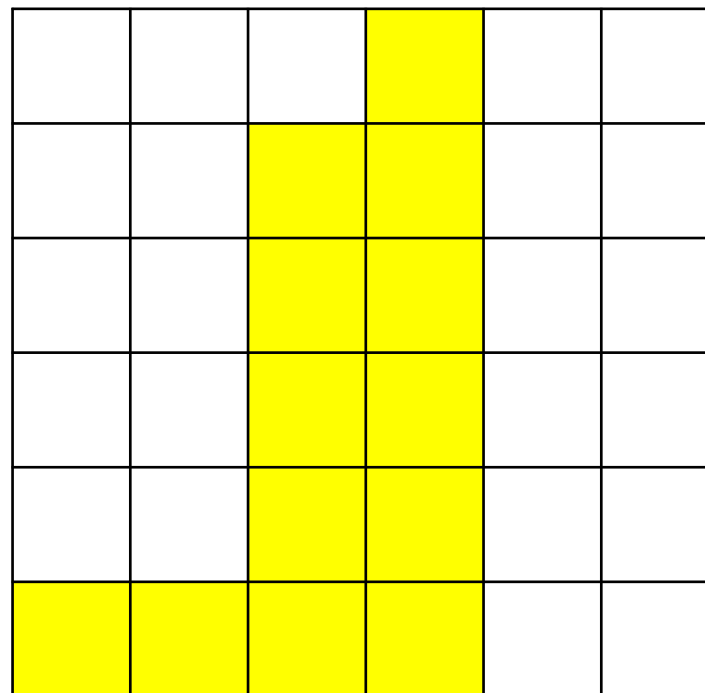




# 維護輪廓

- 有用的區域會形成階梯狀的輪廓
- 只需要記錄轉折點的位置和高度就行了

STK	1	5	6
	0	2	3

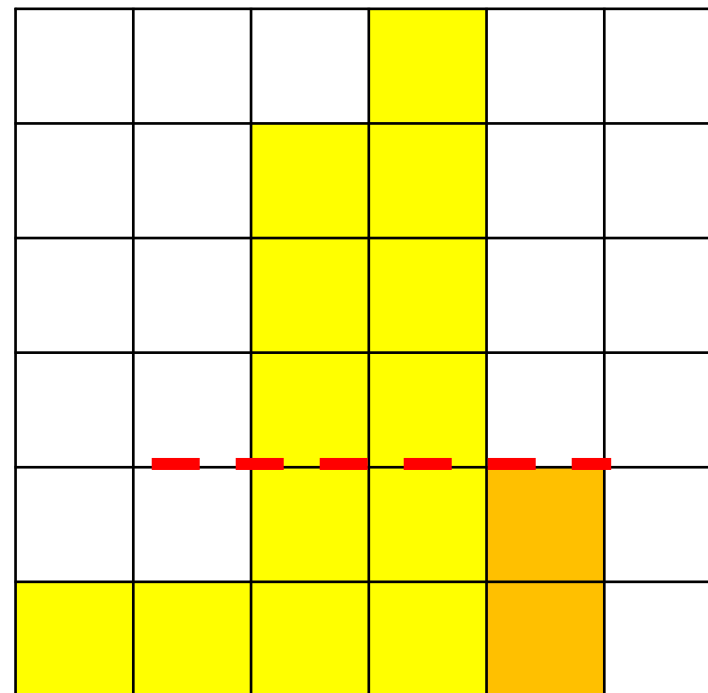


# 維護輪廓

- 有用的區域會形成階梯狀的輪廓
- 只需要記錄轉折點的位置和高度就行了
- 新增下一個長條時  
把要被砍掉的長條 pop 掉  
最後被 pop 的位置會是新的轉折點

STK	1	5	6
	0	2	3

2
4

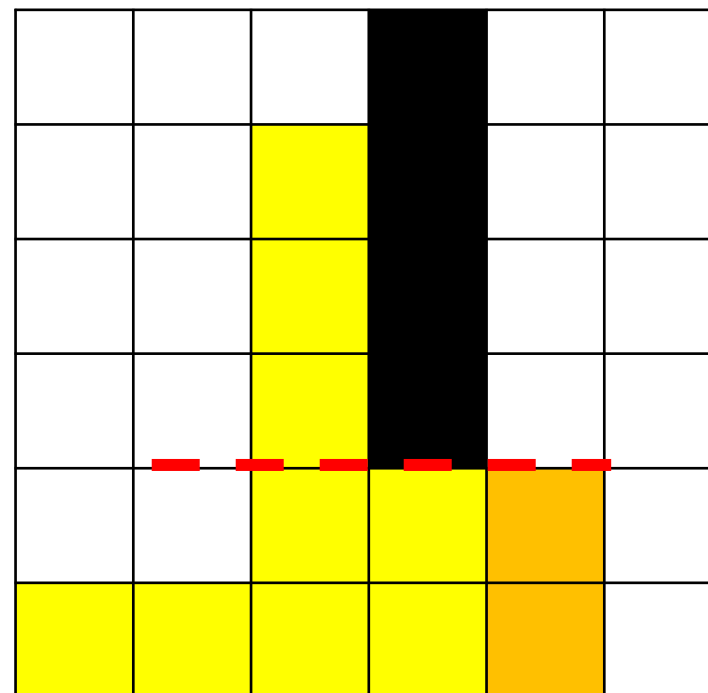


# 維護輪廓

- 有用的區域會形成階梯狀的輪廓
- 只需要記錄轉折點的位置和高度就行了
- 新增下一個長條時  
把要被砍掉的長條 pop 掉  
最後被 pop 的位置會是新的轉折點

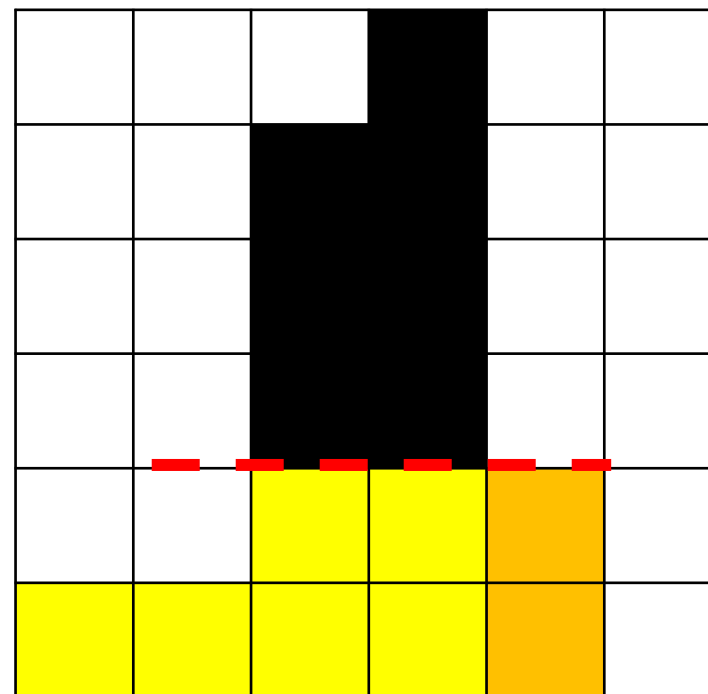
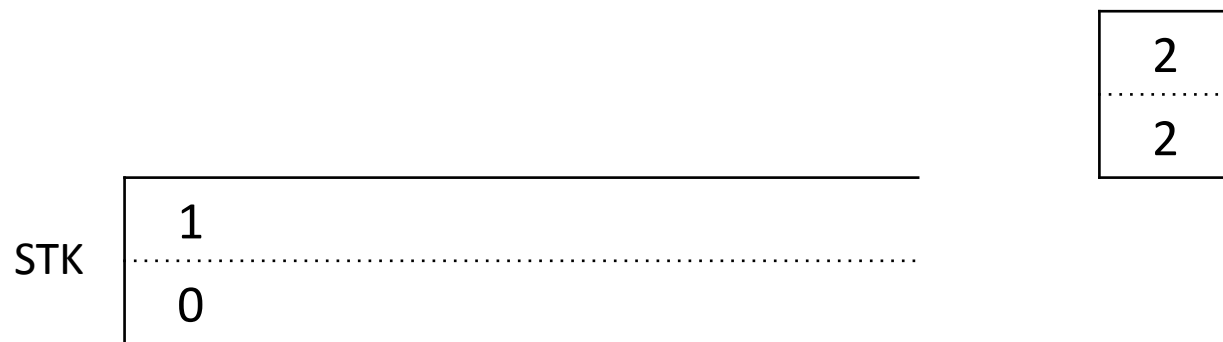
STK	1	5
	0	2

2
3



# 維護輪廓

- 有用的區域會形成階梯狀的輪廓
- 只需要記錄轉折點的位置和高度就行了
- 新增下一個長條時  
把要被砍掉的長條 pop 掉  
最後被 pop 的位置會是新的轉折點






# Stack $O(n)$ 維護輪廓

要考慮兩根一樣長的長條

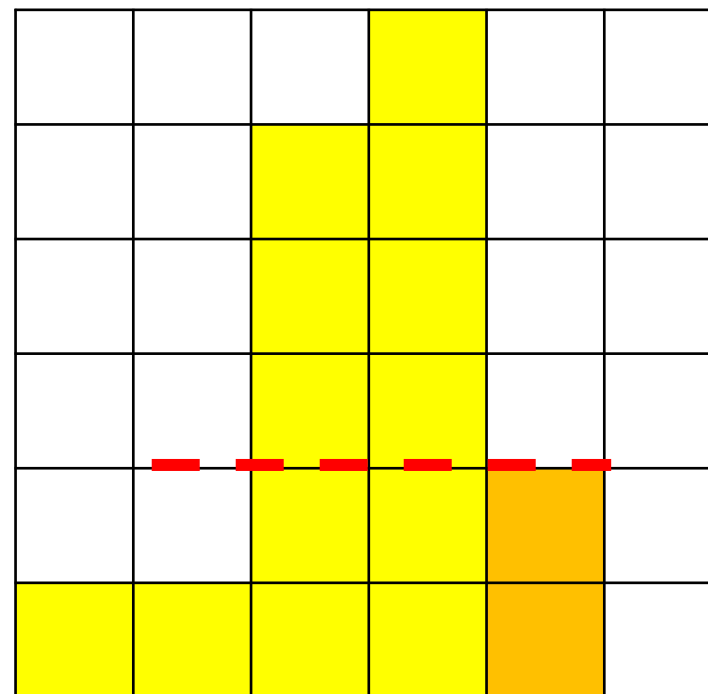
```
long long maxRectangle(vector<int> h) {  
    stack<pair<int, int>> STK;  
    long long ans = 0;  
    for (int i = 0; i < (int)h.size(); ++i) {  
        int corner = i;  
        while (STK.size() && STK.top().first >= h[i]) {  
            corner = STK.top().second;  
            STK.pop();  
        }  
        STK.emplace(h[i], corner);  
    }  
    return ans;  
}
```



# 找出邊界最大化的矩形

- 邊界最大化的矩形一定包含某個轉折點

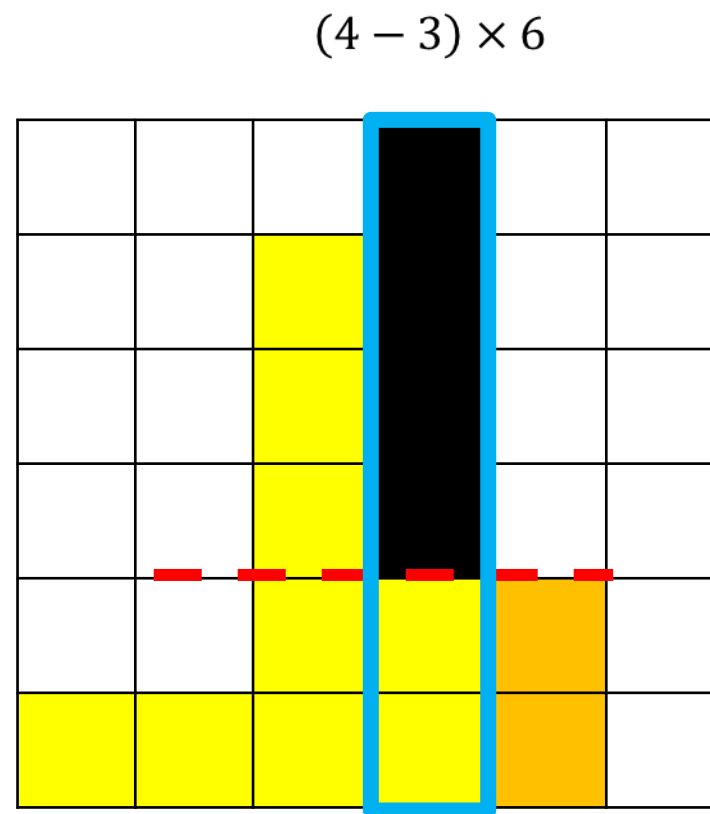
STK	1	5	6
	0	2	3



# 找出邊界最大化的矩形

- 邊界最大化的矩形一定包含某個轉折點
- 轉折點被刪除時就能確定邊界最大化的矩形的位置

STK	1	5
	0	2



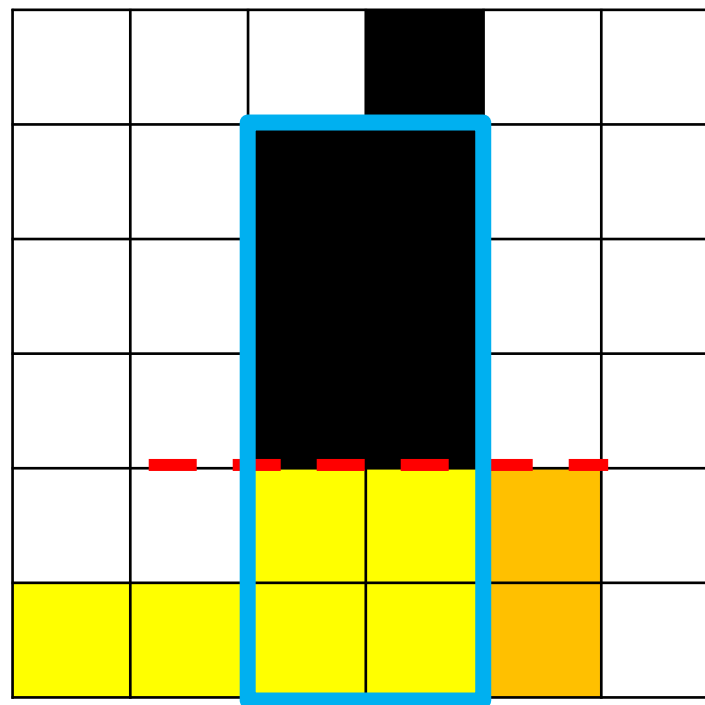


# 找出邊界最大化的矩形

- 邊界最大化的矩形一定包含某個轉折點
- 轉折點被刪除時就能確定邊界最大化的矩形的位置

STK	1
	0

$$(4 - 2) \times 5$$



# 多加兩行就能 $O(n)$ 計算答案

為了最後能刪除所有轉折點

計算當前邊界最大矩形的面積

```
long long maxRectangle(vector<int> h) {  
    h.emplace_back(0);  
    stack<pair<int, int>> STK;  
    long long ans = 0;  
    for (int i = 0; i < (int)h.size(); ++i) {  
        int corner = i;  
        while (STK.size() && STK.top().first >= h[i]) {  
            corner = STK.top().second;  
            ans = max(ans, 1LL * (i - corner) * STK.top().first);  
            STK.pop();  
        }  
        STK.emplace(h[i], corner);  
    }  
    return ans;  
}
```

# Monotonic Queue

保證 deque 中的  
值遞增或遞減

去除永遠不可能  
成為答案的元素

與 stack 最大的差  
別是可以刪掉頭  
部

# Sliding Window

- <https://leetcode.com/problems/sliding-window-maximum/>
- 給你 sliding window 的大小，問你輸入陣列中每個 sliding window 的最大值
- Example
- `nums = [7,6,3,5,-3,-1,3,9]`, `k = 3`
  - `ans = [7,6,5,5,3,9]`

7	6	3	5	-3	-1	3	9	7
7	6	3	5	-3	-1	3	9	6
7	6	3	5	-3	-1	3	9	5
7	6	3	5	-3	-1	3	9	5
7	6	3	5	-3	-1	3	9	3
7	6	3	5	-3	-1	3	9	9

# 考慮使用遞減的 Monotonic Stack

STK

--

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

# 考慮使用遞減的 Monotonic Stack

STK

0




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

# 考慮使用遞減的 Monotonic Stack

STK 

0	1
---	---




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

# 考慮使用遞減的 Monotonic Stack

STK 

0	1	2																	
---	---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--




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



# 考慮使用遞減的 Monotonic Stack

STK 

0	1	3
---	---	---




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

# 考慮使用遞減的 Monotonic Stack

STK

0	1	3	4
---	---	---	---




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

# 考慮使用遞減的 Monotonic Stack

STK

0	1	3	5
---	---	---	---




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

# 考慮使用遞減的 Monotonic Stack

STK

0	1	3	6	
---	---	---	---	--




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

# 考慮使用遞減的 Monotonic Stack

STK 

7							
---	--	--	--	--	--	--	--



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

# 觀察到的性質

- 每個 sliding window 的最大值一定會出現在當前 STK 中
- STK 中的 index 是由小排到大  
nums[index] 由大排到小



用 Binary Search 找出 STK 中  
第一個  $> i - k$  的 index  
就是答案

# 利用 Monotonic Stack + Binary Search

```
vector<int> maxSlidingWindow(vector<int> nums, int k) {  
    vector<int> STK;  
    vector<int> ans;  
    for (int i = 0; i < (int)nums.size(); ++i) {  
        while (STK.size() && nums[STK.back()] < nums[i])  
            STK.pop_back();  
        STK.emplace_back(i);  
        if (i >= k - 1) {  
            auto [L, R] = binarySearch(0, (int)STK.size() - 1,  
                                       [&](int mid) { return STK[mid] <= i - k; });  
            ans.emplace_back(nums[STK[R]]);  
        }  
    }  
    return ans;  
}
```

# 考慮用 Deque 將超出範圍的值刪掉

DQ

---


---

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



# 考慮用 Deque 將超出範圍的值刪掉


DQ 0



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

# 考慮用 Deque 將超出範圍的值刪掉

DQ   0   1




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

# 考慮用 Deque 將超出範圍的值刪掉

DQ   

0	1	2
---	---	---



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

## 考慮用 Deque 將超出範圍的值刪掉

DQ


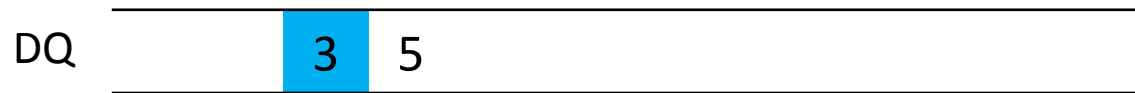
1

3

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



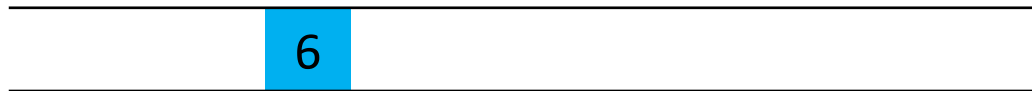
# 考慮用 Deque 將超出範圍的值刪掉



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

# 考慮用 Deque 將超出範圍的值刪掉

DQ

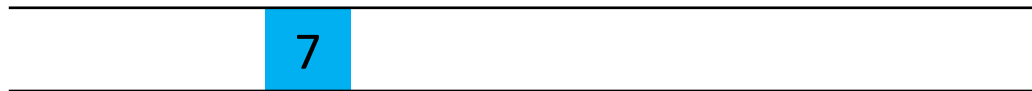


A blue arrow points down to the value 3 in the array at index 6.

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

# 考慮用 Deque 將超出範圍的值刪掉

DQ



A blue arrow points down to the last element of the array, which is 9.

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



# $O(n)$ 四個部分

刪除超過範圍的東西

刪除沒有用的東西

加入當前資料

計算當前答案

根據不同題目性質  
計算當前答案有能出現在  
兩個刪除之間

```
vector<int> maxSlidingWindow(vector<int> &nums, int k) {  
    deque<int> DQ;  
    vector<int> ans;  
    for (int i = 0; i < (int)nums.size(); ++i) {  
        while (DQ.size() && DQ.front() <= i - k)  
            DQ.pop_front();  
  
        while (DQ.size() && nums[DQ.back()] < nums[i])  
            DQ.pop_back();  
  
        DQ.emplace_back(i);  
  
        if (i >= k - 1)  
            ans.emplace_back(nums[DQ.front()]);  
    }  
    return ans;  
}
```