

Computer Programming I Bonus

資工系 114 林振可 (41047029S)

695. Max Area of Island

這是個找地圖中最大陸塊大小的問題。

想法是在遍歷整張地圖時，如果遇到的是陸地，則用 DFS 找該陸地所屬陸塊的大小。

然後輸出其中最大值。

The screenshot shows the LeetCode problem 695. Max Area of Island. The top navigation bar includes 'Explore', 'Problems', 'Interview', 'Contest', 'Discuss', and 'Store'. The current tab is 'Submissions'. On the left, there's a 'Success' section with details: Runtime: 8 ms, faster than 100.00% of C online submissions for Max Area of Island; Memory Usage: 6.9 MB, less than 37.93% of C online submissions for Max Area of Island. Below that is a 'Next challenges:' section with links to other problems. A 'Show off your acceptance:' button is followed by social sharing icons for Facebook, Twitter, and LinkedIn. The main area displays the C++ code for the solution:

```
i C Autocomplete
1 int maxAreaOfIsland(int** grid, int gridSize, int* gridColSize) {
2     int m = gridSize, n = gridColSize[0];
3
4     int find(int x, int y) {
5         // 如果是 (x, y) 是海洋或是已經找過了，就不要繼續
6         if (grid[y][x] == 0 || grid[y][x] == 2) return 0;
7
8         // area 是這一個陸塊的大小
9         int area = 1;
10
11        // 紀錄這塊地已找過
12        grid[y][x] = 2;
13
14        // 如果右邊不是邊界，找右邊
15        if (x + 1 < n) area += find(x + 1, y);
16        // 如果左邊不是邊界，找左邊
17        if (x > 0) area += find(x - 1, y);
18        // 如果下面是邊界，找下面
19        if (y + 1 < m) area += find(x, y + 1);
20        // 如果上面不是邊界，找上面
21        if (y > 0) area += find(x, y - 1);
22
23        // 回傳該陸塊總大小
24        return area;
25    }
26
27
28    int max = 0;
29
30    // 遍歷所有位置
31    for (int i = 0; i < m; i++) {
32        for (int j = 0; j < n; j++) {
33            // 如果是陸地，則找該陸地所屬陸塊大小
34            if (grid[i][j] == 1) {
35                int area = find(j, i);
36                max = area > max ? area : max;
37            }
38        }
39    }
40}
```

The right sidebar shows the user profile of JacobLinCool with links to 'My List', 'My Playground', 'Notebook', 'Submissions', 'Sessions', 'Progress', 'Points', 'Subscription', 'Orders', and 'Sign out'. At the bottom, there are buttons for 'Run Code', 'Submit', and navigation links like 'Problems', 'Pick One', 'Prev', 'Next', 'Console', and 'Contribute'.

```
1 int maxAreaOfIsland(int** grid, int gridSize, int* gridColSize) {
2     int m = gridSize, n = gridColSize[0];
3
4     int find(int x, int y) {
5         // 如果是 (x, y) 是海洋或是已經找過了，就不要繼續
6         if (grid[y][x] == 0 || grid[y][x] == 2) return 0;
7
8         // area 是這一個陸塊的大小
9         int area = 1;
10
11        // 紀錄這塊地已找過
12        grid[y][x] = 2;
13
14        // 如果右邊不是邊界，找右邊
15        if (x + 1 < n) area += find(x + 1, y);
16        // 如果左邊不是邊界，找左邊
17        if (x > 0) area += find(x - 1, y);
18        // 如果下面不是邊界，找下面
19        if (y + 1 < m) area += find(x, y + 1);
20        // 如果上面不是邊界，找上面
21        if (y > 0) area += find(x, y - 1);
22
23        // 回傳該陸塊總大小
24        return area;
25    }
26
27    int max = 0;
28
29    // 遍歷所有位置
30    for (int i = 0; i < m; i++) {
31        for (int j = 0; j < n; j++) {
32            // 如果是陸地，則找該陸地所屬陸塊大小
33            if (grid[i][j] == 1) {
34                int area = find(j, i);
35                max = area > max ? area : max;
36            }
37        }
38    }
39
40    return max;
41 }
```

1465. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

這題看完就會發現你其實只要分別找高的最大值與寬的最大值就好了。

因為最大那塊大小就是高的最大值與寬的最大值的乘積。

找最大值的方法，以縱向的高為例：

我們就先將所有切割線做遞增排序，可以得到如下關係

- 上方邊界
- 第一水平切割線
- 第二水平切割線
- ...
- 倒數第二水平切割線
- 倒數第一水平切割線
- 下方邊界

遍歷跑過去找相鄰兩個距離的最大值就可以得到答案了。

而橫向的寬也是如此。

最後再將高的最大值與寬的最大值乘起來取模 $10^9 + 7$ 就好了。

The screenshot shows the LeetCode platform interface for problem 1465. The top navigation bar includes 'LeetCode', 'Explore', 'Problems', 'Interview', 'Contest', 'Discuss', and 'Store'. The right sidebar shows the user's profile: 'JacobLinCool', 'My List', 'My Playground', 'Notebook', 'Submissions', 'Sessions', 'Progress', 'Points', 'Subscription', 'Orders', and 'Sign out'. The main area displays the following information:

- Description:** Success, Details >
- Runtime:** 56 ms, faster than 100.00% of C online submissions for Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts.
- Memory Usage:** 12.2 MB, less than 100.00% of C online submissions for Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts.
- Next challenges:** Wiggle Sort II, Two Sum Less Than K, Restore the Array From Adjacent Pairs.
- Show off your acceptance:** [f](#), [t](#), [in](#)
- Table of Submissions:** Shows the user's submission history with columns: Time Submitted, Status, Runtime, Memory, Language. Submissions include:
 - 12/26/2021 22:37 Accepted 56 ms 12.2 MB c
 - 12/26/2021 22:35 Runtime Error N/A N/A c
 - 06/04/2021 13:44 Accepted 144 ms 47.1 MB javascript
 - 06/04/2021 13:44 Accepted 140 ms 48.1 MB javascript
 - 06/04/2021 13:44 Wrong Answer N/A N/A javascript
- Code Editor:** Displays the C++ solution code for calculating the maximum area of a piece of cake after horizontal and vertical cuts.
- Bottom Navigation:** Includes buttons for 'Problems', 'Pick One', 'Prev', '1465/2122', 'Next', 'Console', 'Contribute i', 'Run Code', and 'Submit'.

```
1 int compare (const int *a, const int *b) {
2     return (*a - *b);
3 }
4
5 int maxArea(int h, int w, int horizontalCuts[], int horizontalCutsSize,
6 int verticalCuts[], int verticalCutsSize){
7     // 先把切割線都做遞增排序
8     qsort(horizontalCuts, horizontalCutsSize, sizeof(int), compare);
9     qsort(verticalCuts, verticalCutsSize, sizeof(int), compare);
10
11     // 紀錄最大高度以及最大寬度，初始值設為第一條切割線到相鄰邊界的距離
12     int64_t max_height = horizontalCuts[0], max_width =
13 verticalCuts[0];
14
15     // 逐步看兩相鄰切割線距離，找最大高度
16     for(int i = 1; i < horizontalCutsSize; i++) {
17         int64_t height = horizontalCuts[i] - horizontalCuts[i - 1];
18         max_height = height > max_height ? height : max_height;
19     }
20     // 也要看看最後一條切割線到相鄰邊界的距離
21     int64_t height = h - horizontalCuts[horizontalCutsSize - 1];
22     max_height = height > max_height ? height : max_height;
23
24     // 逐步看兩相鄰切割線距離，找最大寬度
25     for(int i = 1; i < verticalCutsSize; i++) {
26         int64_t width = verticalCuts[i] - verticalCuts[i - 1];
27         max_width = width > max_width ? width : max_width;
28     }
29     // 看看最後一條切割線到相鄰邊界的距離
30     int64_t width = w - verticalCuts[verticalCutsSize - 1];
31     max_width = width > max_width ? width : max_width;
32
33     // 輸出最大塊面積
34     return (int)((max_height * max_width) % (int64_t)1000000007);
35 }
```

1690. Stone Game VII

這題可能會想到一個用 DFS 的作法。

但那樣遞迴下去找差會跑到 TLE。

所以我們用動態規劃來處理，因為每多一個元素的情況，前一個的差就會反轉過來。

我們有個二維 DP 陣列代表最佳差。

左下三角代表頭比尾後面，所以我們不會用到。

主對角線則是頭尾位置相同，只有單一元素的情況，所以差也為 0。

我們只需要處理右上三角，增加元素的同時扣除前一個的差。

The screenshot shows the LeetCode platform interface for problem 1690. Stone Game VII. The top navigation bar includes 'Explore', 'Problems', 'Interview', 'Contest', 'Discuss', and 'Store'. The current tab is 'Solution'.

Description: Success Details >

Runtime: 52 ms, faster than 100.00% of C online submissions for Stone Game VII.

Memory Usage: 10.4 MB, less than 100.00% of C online submissions for Stone Game VII.

Next challenges:

- Stone Game I
- Stone Game II
- Stone Game III
- Stone Game IV
- Stone Game V
- Stone Game VI
- Maximum Score from Performing Multiplication Operations
- Stone Game VIII
- Stone Game IX

Show off your acceptance: [f](#) [t](#) [l](#)

Time Submitted	Status	Runtime	Memory	Language
12/27/2021 01:33	Accepted	52 ms	10.4 MB	c
06/11/2021 16:09	Accepted	592 ms	64.6 MB	javascript

Code Editor:

```
i c    Autocomplete
Success Details >
Runtime: 52 ms, faster than 100.00% of C online submissions for Stone Game VII.
Memory Usage: 10.4 MB, less than 100.00% of C online submissions for Stone Game VII.

int stoneGameVII(int stones[], int stonesSize){
    // dp[a][b] 代表石頭第一個位置是 a 最後一個位置是 b 的情況
    int dp[stonesSize][stonesSize];
    memset(dp, 0, sizeof(dp));
    for(int i = stonesSize - 1; i >= 0; i--) {
        // sum 就是區段的分數總和
        int sum = stones[i];
        for(int j = i + 1; j < stonesSize; j++) {
            sum += stones[j];
            // 移除第一個的差，多了一個元素，所以把上次的反轉
            int remove_first = sum - stones[i] - dp[i];
            // 移除最後一個的差，多了一個元素，所以把上次的反轉
            int remove_last = sum - stones[j] - dp[i];
            // Alex 必定會選較大的那個
            dp[i][j] = remove_first > remove_last ? remove_first : remove_last;
        }
        // 回傳全區段的情形下的差
        return dp[0][stonesSize - 1];
    }
}
```

User Profile: JacobLinCool

- My List
- My Playground
- Notebook
- Submissions
- Sessions
- Progress
- Points
- Subscription
- Orders
- Sign out

Console ▾ Contribute i Run Code ▾ Submit

```
1 int stoneGameVII(int stones[], int stonesSize){  
2     // dp[a][b] 代表石頭第一個位置是 a 最後一個位置是 b 的差  
3     int dp[stonesSize][stonesSize];  
4     memset(dp, 0, sizeof(dp));  
5  
6     for(int i = stonesSize - 1; i >= 0; i--) {  
7         // sum 就是區段的分數總和  
8         int sum = stones[i];  
9         for(int j = i + 1; j < stonesSize; j++) {  
10             sum += stones[j];  
11  
12             // 移除第一個的差，多了一個元素，所以把上次的反轉扣除  
13             int remove_first = sum - stones[i] - dp[i + 1][j];  
14             // 移除最後一個的差，多了一個元素，所以把上次的反轉扣除  
15             int remove_last = sum - stones[j] - dp[i][j - 1];  
16  
17             // Alex 必定會選較大的那個  
18             dp[i][j] = remove_first > remove_last ? remove_first :  
19             remove_last;  
20         }  
21     }  
22  
23     // 回傳全區段的情形下的差  
24     return dp[0][stonesSize - 1];  
}
```

729. My Calendar I

這題是很簡單的應用題，只要檢查時段碰撞就好了。

分為三種情況，同時只有其一狀況會發生：

1. 新事件開始時間早於舊事件開始時間
2. 新事件開始時間在舊事件區間內
3. 新事件開始時間晚於舊事件結束時間

第 1 種情況，如果新事件在舊事件開始後結束就會撞到。

第 2 種情況，一定會撞到。

第 3 種情況，一定不會撞到。

所以我們只需擋掉前兩種狀況。

通過所有測試的話，就把新事件加入事件列表。

The screenshot shows the LeetCode platform interface for problem 729. The top navigation bar includes 'Explore', 'Problems', 'Interview', 'Contest', 'Discuss', and 'Store'. The current tab is 'Solution'.

Success Details >

Runtime: 104 ms, faster than 28.57% of C online submissions for My Calendar I.

Memory Usage: 22.3 MB, less than 85.71% of C online submissions for My Calendar I.

Next challenges:

[My Calendar II](#) [My Calendar III](#)

Show off your acceptance: [f](#) [t](#) [in](#)

Time Submitted	Status	Runtime	Memory	Language
12/27/2021 01:46	Accepted	104 ms	22.3 MB	C
06/10/2021 16:38	Accepted	164 ms	47.4 MB	javascript
06/10/2021 16:32	Accepted	376 ms	45.4 MB	javascript
06/10/2021 16:31	Accepted	420 ms	47.3 MB	javascript
06/10/2021 16:31	Accepted	192 ms	46.9 MB	javascript
06/10/2021 16:26	Accepted	200 ms	46.8 MB	javascript

Code Editor:

```
i C Autocomplete
1 typedef struct Event {
2     int start;
3     int end;
4 } Event;
5
6 typedef struct MyCalendar {
7     Event *events;
8     int size;
9 } MyCalendar;
10
11 MyCalendar* myCalendarCreate() {
12     MyCalendar *calendar = malloc(sizeof(MyCalendar));
13     calendar->events = malloc(sizeof(Event) * 1000);
14     calendar->size = 0;
15     return calendar;
16 }
17
18 bool myCalendarBook(MyCalendar* calendar, int start,
19 // 把已經建立的事件都做一次碰撞測試
20 for(int i = 0; i < calendar->size; i++) {
21     Event event = calendar->events[i];
22     // 如果新事件開始比舊事件開始早，但結束在舊事件開始後
23     if (start < event.start && end > event.end)
24         return false;
25     // 如果新事件開始在舊事件時間範圍內，也一定會撞到
26     else if (start >= event.start && start < event.end)
27         return false;
28     }
29
30
31
32     // 如果沒有撞到，就把新事件加入
33     Event new_event = { start, end };
34     calendar->events[calendar->size++] = new_event;
35     return true;
36 }
37
```

User Profile: JacobLinCool

- My List
- My Playground
- Notebook
- Submissions
- Sessions
- Progress
- Points
- Subscription
- Orders
- Sign out

Console Contribute i Run Code ^ Submit

```
1  typedef struct Event {
2      int start;
3      int end;
4  } Event;
5
6  typedef struct MyCalendar {
7      Event *events;
8      int size;
9  } MyCalendar;
10
11 MyCalendar* myCalendarCreate() {
12     MyCalendar *calendar = malloc(sizeof(MyCalendar));
13     calendar->events = malloc(sizeof(Event) * 1000);
14     calendar->size = 0;
15     return calendar;
16 }
17
18 bool myCalendarBook(MyCalendar* calendar, int start, int end) {
19     // 把已經建立的事件都做一次碰撞測試
20     for(int i = 0; i < calendar->size; i++) {
21         Event event = calendar->events[i];
22         // 如果新事件開始比舊事件開始早，但結束在舊事件開始後，一定會撞到
23         if (start < event.start && end > event.start) {
24             return false;
25         }
26         // 如果新事件開始在舊事件時間範圍內，也一定會撞到
27         else if (start >= event.start && start < event.end) {
28             return false;
29         }
30     }
31
32     // 如果沒有撞到，就把新事件加入
33     Event new_event = { start, end };
34     calendar->events[calendar->size++] = new_event;
35     return true;
36 }
37
38 void myCalendarFree(MyCalendar* calendar) {
39     free(calendar->events);
40     free(calendar);
41 }
```

746. Min Cost Climbing Stairs

這個階梯問題最一般的方法應該會是用 DP 來做。

因為到第 x 階所要花的總 cost 就會是到第 $x - 1$ 階（最後再跨 1 步到 x ）或到第 $x - 2$ 階（最後再跨 2 步到 x ）兩者之間的較小總 cost 再加上第 x 階自己的 cost。

那用一個 DP 陣列，初始放第一階及第二階的 cost，跑上去答案就會出來了。

再仔細想想，其實連 DP 陣列都不用，因為每次只會需要前兩階的總 cost。

可以用像滾輪的方式，類似像 DP 概念一樣慢慢滾上去，就可以算出來了。

The screenshot shows the LeetCode platform interface for problem 746. The top navigation bar includes 'Explore', 'Problems', 'Interview', 'Contest', 'Discuss', and 'Store'. The current problem page has tabs for 'Description', 'Solution', 'Discuss (999+)', and 'Submissions'. The 'Success' status is shown with a 4 ms runtime and 5.7 MB memory usage.

The code editor on the right contains the following C code for 'minCostClimbingStairs':

```
int minCostClimbingStairs(int cost[], int costSize){  
    // 差一步「之前」的總 cost 跟差兩步「之前」的總 cost，起  
    int one = 0;  
    two = 0;  
    // 滾上去，記得要滾到 costSize  
    for (int i = 2; i <= costSize; i++) {  
        // 每次往上一層，差一步「之前」的總 cost 會變成差兩步  
        int new_two = one;  
        // 而新的差一步「之前」的總 cost 則會是原本差一步「之  
        // 差兩步「之前」的總 cost + 差兩步的 cost，兩者較小值  
        one = one + cost[i - 1] < two + cost[i - 2] ?  
            - 2];  
        two = new_two;  
    }  
    // 最後輸出的其實是到 n + 1 階差一步「之前」的總 cost  
    return one;  
}
```

The submission history table shows the following data:

Time Submitted	Status	Runtime	Memory	Language
12/27/2021 02:38	Accepted	4 ms	5.7 MB	c
06/07/2021 18:07	Accepted	84 ms	41 MB	javascript
06/07/2021 18:06	Accepted	116 ms	40.3 MB	javascript
06/07/2021 18:05	Accepted	92 ms	40.3 MB	javascript
06/07/2021 18:04	Accepted	164 ms	40.6 MB	javascript
06/07/2021 18:04	Accepted	96 ms	40.3 MB	javascript

The right sidebar displays the user's profile for JacobLinCool, including sections for 'My List', 'My Playground', 'Notebook', 'Submissions', 'Sessions', 'Progress', 'Points', 'Subscription', 'Orders', and 'Sign out'.

```
1 int minCostClimbingStairs(int cost[], int costSize){  
2     // 差一步「之前」的總 cost 跟差兩步「之前」的總 cost，起始 0 意義是第零階的  
3     cost  
4         int one = 0,  
5             two = 0;  
6  
7     // 滾上去，記得要滾到 costSize  
8     for (int i = 2; i <= costSize; i++) {  
9         // 每次往上一層，差一步「之前」的總 cost 會變成差兩步「之前」的總 cost  
10        int new_two = one;  
11        // 而新的差一步「之前」的總 cost 則會是原本差一步「之前」的總 cost + 差一  
12        // 步的 cost 與原本差兩步「之前」的總 cost + 差兩步的 cost，兩者較小值  
13        one = one + cost[i - 1] < two + cost[i - 2] ? one + cost[i - 1]  
14        : two + cost[i - 2];  
15        two = new_two;  
16    }  
17  
18    // 最後輸出的其實是到 n + 1 階差一步「之前」的總 cost  
19    return one;  
}
```

9. Palindrome Number

因為依照迴文數的定義，其反轉後也必會是原本的數。

所以我們先製造一個與 x 前後相反的數 $reversed$ ，再比較兩者是否相等就可得知結果。

The screenshot shows the LeetCode submission page for the Palindrome Number problem. The code is written in C and passes all test cases.

```
1+ bool isPalindrome(int x){  
2     // x 是負數，直接 false  
3     if(x < 0) return false;  
4  
5     // y 用來存 x 的反轉數  
6     int64_t reversed = 0;  
7  
8     // 從最後一位開始，每次將 reversed 左推後加上該位之值  
9     for(int64_t num = x; num > 0; num /= 10) {  
10         reversed = reversed * 10 + num % 10;  
11     }  
12  
13     // 比較兩者是否相等  
14     return (int64_t)x == reversed;  
15 }
```

The submission details are as follows:

Time Submitted	Status	Runtime	Memory	Language
12/27/2021 03:34	Accepted	8 ms	6.1 MB	c
12/27/2021 03:26	Accepted	12 ms	6.1 MB	c
12/27/2021 03:23	Accepted	8 ms	5.8 MB	c
12/27/2021 03:22	Runtime Error	N/A	N/A	c

The right sidebar shows the user's profile: JacobLinCool, with links to My List, My Playground, Notebook, Submissions, Sessions, Progress, Points, Subscription, Orders, and Sign out.

Problems | Pick One | < Prev | 9/2122 | Next > | Console | Contribute i | Run Code | Submit

```
1  bool isPalindrome(int x){  
2      // x 是負數，直接 false  
3      if(x < 0) return false;  
4  
5      // y 用來存 x 的反轉數  
6      int64_t reversed = 0;  
7  
8      // 從最後一位開始，每次將 reversed 左推後加上該位之值  
9      for(int64_t num = x; num > 0; num /= 10) {  
10          reversed = reversed * 10 + num % 10;  
11      }  
12  
13      // 比較兩者是否相等  
14      return (int64_t)x == reversed;  
15 }
```

1009. Complement of Base 10 Integer

這題要把二進位數字每位都翻轉過來 ($0 \rightarrow 1, 1 \rightarrow 0$)。

但要忽略小數字前面的 0，像 $00\dots00101$ 應該要變成 $00\dots00010$ ，而不是 $11\dots11010$ 。所以不能直接用 $n \text{ XOR } 0xFFFFFFFF$ ，但我們可以分別就每一位進行 XOR 直到有數字的最高位。

XOR 表：

$0 \wedge 1 = 1$
$1 \wedge 1 = 0$
$0 \wedge 0 = 0$
$1 \wedge 0 = 1$

我們每次做 XOR 時只做一位，其他位就是 $n \wedge 0 = n$ 維持原狀。

The screenshot shows the LeetCode problem page for "Complement of Base 10 Integer". The code editor displays the following C code:

```
i C * Autocomplete
1 int bitwiseComplement(int n) {
2     // 如果 n 為 0，直接回傳 1
3     if(n == 0) return 1;
4
5     int flipped = n;
6
7     // 從最尾端開始遍歷，每次翻一個，其餘維持原狀
8     for(int64_t i = 1; i <= n; i <= 1) {
9         flipped = flipped ^ i;
10    }
11
12    return flipped;
13 }
```

The right sidebar shows the user's profile: JacobLinCool, with links to My List, My Playground, Notebook, Submissions, Sessions, Progress, Points, Subscription, Orders, and Sign out.

The bottom navigation bar includes: Problems, Pick One, Prev, 1009/2122, Next, Console, Contribute i, Run Code, and Submit.

```
1 int bitwiseComplement(int n) {
2     // 如果 n 為 0，直接回傳 1
3     if(n == 0) return 1;
4
5     int flipped = n;
6
7     // 從最尾端開始遍歷，每次翻一個，其餘維持原狀
8     for(int64_t i = 1; i <= n; i <= 1) {
9         flipped = flipped ^ i;
10    }
11
12    return flipped;
13 }
```

26. Remove Duplicates from Sorted Array

移除遞增數列中的重複數字，使其嚴格遞增。

我們紀錄兩個位置，一個是保證嚴格遞增狀態的尾端，另一個是處理的前端：

```
[1, 2, 2, 3, 4, 4, 4, 5]  
  ^c ^
```

```
[1, 2, 2, 3, 4, 4, 4, 5]  
  ^c ^
```

```
[1, 2, 2, 3, 4, 4, 4, 5]  
  ^c      ^
```

```
[1, 2, 3, 3, 4, 4, 4, 5]  
  ^c      ^
```

```
[1, 2, 3, 4, 4, 4, 4, 5]  
  ^c      ^
```

```
[1, 2, 3, 4, 4, 4, 4, 5]  
  ^c      ^
```

```
[1, 2, 3, 4, 5, 4, 4, 5]  
  ^c           ^
```

```
result: [1, 2, 3, 4, 5], length: position of c + 1
```

從前面到後面，線性處理就好了。

記得如果 `numsSize` 本來就是 0 要回傳 0。

[Description](#) [Solution](#) [Discuss \(999+\)](#) [Submissions](#)

Success Details >

Runtime: 8 ms, faster than 99.06% of C online submissions for Remove Duplicates from Sorted Array.

Memory Usage: 7.4 MB, less than 82.83% of C online submissions for Remove Duplicates from Sorted Array.

Next challenges:

[Remove Element](#) [Remove Duplicates from Sorted Array II](#)

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/27/2021 18:47	Accepted	8 ms	7.4 MB	c
12/27/2021 18:46	Accepted	12 ms	7.8 MB	c
12/27/2021 13:13	Accepted	12 ms	7.7 MB	c

i C Autocomplete

```
1 int removeDuplicates(int nums[], int numsSize) {
2     // 當前要與後面比較的 index
3     int current = 0;
4
5     // 從第二個開始，與當前的值比較
6     for(int i = 1; i < numsSize; i++) {
7         if(nums[i] != nums[current]) {
8             nums[++current] = nums[i];
9         }
10    }
11
12    return numsSize ? current + 1 : 0;
13 }
```

- JacobLinCool >
- My List
- My Playground
- Notebook
- Submissions
- Sessions
- Progress
- Points
- Subscription
- Orders
- Sign out

[Problems](#) [X Pick One](#) [◀ Prev](#) 26/2122 [Next ▶](#) [Console](#) [Contribute i](#) [▶ Run Code ^](#) [Submit](#)

```
1 int removeDuplicates(int nums[], int numsSize) {
2     // 當前要與後面比較的 index
3     int current = 0;
4
5     // 從第二個開始，與當前的值比較
6     for(int i = 1; i < numsSize; i++) {
7         if(nums[i] != nums[current]) {
8             nums[++current] = nums[i];
9         }
10    }
11
12    return numsSize ? current + 1 : 0;
13 }
```

28. Implement strStr()

實作出 c 裡面的 strstr 函式，也就是找第一個 substring 的位置。

我們用編譯器高度優化的 memcmp 函式從頭進行比對。

```
int memcmp ( void* ptr1, void* ptr2, size_t size );
```

因為 char* 跟 void* 的大小都是 1 byte，所以 size 就是 needle 的長度。

The screenshot shows the LeetCode platform interface. On the left, there's a navigation bar with 'Explore', 'Problems', 'Interview', 'Contest', 'Discuss', and 'Store'. Below it, a 'Success' message indicates a runtime of 12 ms and memory usage of 6 MB. A table lists five previous submissions, all accepted. The main area contains the C code for the strStr function, which uses the memcmp function to compare substrings. On the right, a sidebar shows the user's profile with sections for 'My List', 'My Playground', 'Notebook', 'Submissions', 'Sessions', 'Progress', 'Points', 'Subscription', 'Orders', and 'Sign out'.

Time Submitted	Status	Runtime	Memory	Language
12/27/2021 21:22	Accepted	12 ms	6 MB	c
12/27/2021 21:13	Time Limit Exceeded	N/A	N/A	c
12/27/2021 21:10	Runtime Error	N/A	N/A	c
12/27/2021 20:11	Time Limit Exceeded	N/A	N/A	c

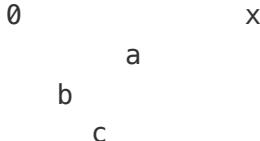
```
i C Autocomplete
1 int strStr(char haystack[], char needle[]) {
2     // 如果 needle 為空字串，直接回傳 0
3     if (needle[0] == '\0') return 0;
4
5     // 計算兩字串長度
6     int haystack_length = strlen(haystack);
7     int needle_length = strlen(needle);
8
9     // haystack 從頭開始比對，直到剩餘長度不足配對完整 needle
10    for(int i = 0; i < haystack_length - needle_length + 1; i++) {
11        // 用 memcmp 比較
12        if(memcmp(haystack + i, needle, needle_length) == 0) {
13            return i;
14        }
15    }
16    return -1;
17 }
```

```
1 int strStr(char haystack[], char needle[]) {
2     // 如果 needle 為空字串，直接回傳 0
3     if (needle[0] == '\0') return 0;
4
5     // 計算兩字串長度
6     int haystack_length = strlen(haystack);
7     int needle_length = strlen(needle);
8
9     // haystack 從頭開始比對，直到剩餘長度不足配對完整 needle
10    for(int i = 0; i < haystack_length - needle_length + 1; i++) {
11        // 用 memcmp 比較
12        if(memcmp(haystack + i, needle, needle_length) == 0) {
13            return i;
14        }
15    }
16    return -1;
17 }
```

69. Sqrt(x)

我們用二分搜尋法去找 x 的平方根，左界取 0，右界取 x 。

類似像這樣 ($a \rightarrow b \rightarrow c$) :



LeetCode Submission Details for mySqrt:

- Description
- Solution
- Discuss (999+)
- Submissions
- i C Autocomplete

Success Details >

Runtime: 4 ms, faster than 77.02% of C online submissions for Sqrt(x).

Memory Usage: 5.5 MB, less than 93.17% of C online submissions for Sqrt(x).

Next challenges:

Pow(x, n) Valid Perfect Square

Show off your acceptance: [f](#) [t](#) [in](#)

Time Submitted	Status	Runtime	Memory	Language
12/27/2021 23:46	Accepted	4 ms	5.5 MB	c
12/27/2021 23:44	Accepted	16 ms	5.8 MB	c
12/27/2021 23:43	Runtime Error	N/A	N/A	c

Console Contribute i Run Code Submit

Code (C):

```
int mySqrt(int64_t x) {
    // 左界為 l, 右界為 r
    int64_t l = 0, r = x;
    // 二分搜尋，直到 l 超過 r
    while(l <= r) {
        // 取中間值
        int64_t m = l + (r - l) / 2;
        // 如果測試值比 x 小則往上找，反之往下找
        if(m * m <= x) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    return r;
}
```

```
int mySqrt(int64_t x) {
    // 左界為 l, 右界為 r
    int64_t l = 0, r = x;
    // 二分搜尋，直到 l 超過 r
    while(l <= r) {
        // 取中間值
        int64_t m = l + (r - l) / 2;
        // 如果測試值比 x 小則往上找，反之往下找
        if(m * m <= x) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    return r;
}
```

