VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Cấu Trúc Dữ Liệu và Giải Thuật (DSA)

DSA1 - HK242

Ôn KTLT - Array

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 1/2025

Mục lục

1 Hàm missingNumber **2**

2 Hàm singleNumber **3**

3 Hàm findSecondSmallest **4**

4 Hàm maxSubArray **5**

5 Hàm isMountainArray **6**

6 Hàm findMaxSumSubarray **7**

7 Hàm maxAreaOfIsland **8**

8 Hàm numberOfClosedIslands **9**

1 Hàm missingNumber

Bài Toán: Bài toán missingNumber yêu cầu bạn tìm số thiếu trong một mảng chứa các số nguyên không âm từ 0 đến n. Cụ thể, bạn được cung cấp một mảng nums có kích thước size, trong đó size = n. Mảng này chứa các số duy nhất từ 0 đến n trừ một số bị thiếu. Nhiệm vụ của bạn là xác định và trả về số bị thiếu đó.

Code: KTLT_ARRAY/src/findMaxSumSubarray.cpp

```
1 int missingNumber(int nums[], int size);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/missingNumberXX.cpp

Gợi ý để giải quyết bài toán

1. **Công Thức Tổng Số Học:** Tính tổng lý thuyết từ `0` đến `n` và trừ đi tổng các phần tử trong mảng để tìm số thiếu.
2. **Phép XOR:** Sử dụng phép XOR để loại bỏ các số xuất hiện trong cả chỉ số và mảng, kết quả còn lại là số bị thiếu.
3. **Sử dụng Bộ Nhớ Phụ (Hash Set/Bộ Đánh Dấu):** Đánh dấu sự hiện diện của từng số trong mảng bằng một cấu trúc dữ liệu phụ và tìm số chưa được đánh dấu.
4. **Đặt Giá Trị Vào Chỉ Số Tương Ứng (Index Mapping):** Sắp xếp các số trong mảng sao cho mỗi số được đặt vào vị trí chỉ số tương ứng và xác định số thiếu dựa trên sự không khớp.

Phương Pháp	Time	Space	Ưu Điểm	Nhược Điểm
1. Công Thức Tổng Số Học	O(n)	O(1)	Đơn giản, hiệu quả	Có thể gặp vấn đề về độ chính xác
2. Phép XOR	O(n)	O(1)	Không gặp vấn đề về độ chính xác	Ít trực quan hơn
3. Sắp Xếp Mảng	O(nlogn)	O(1 | n)	Dễ hiểu, mở rộng dễ dàng	Thời gian chậm hơn, cần thay đổi mảng
4. Đặt Giá Trị Vào Chỉ Số Tương Ứng	O(n)	O(1)	Thời gian hiệu quả, không cần bộ nhớ phụ	Phức tạp hơn trong triển khai, thay đổi mảng gốc

Bảng 1: *Tóm Tắt Các Phương Pháp Giải Bài Toán* `missingNumber`

2 Hàm singleNumber

Bài Toán: Bài toán singleNumber yêu cầu bạn tìm số duy nhất xuất hiện một lần trong một mảng, trong khi tất cả các số còn lại xuất hiện đúng hai lần. Cụ thể, bạn được cung cấp một mảng nums chứa các số nguyên không âm, trong đó chỉ có một số xuất hiện một lần và các số còn lại xuất hiện hai lần. Nhiệm vụ của bạn là xác định và trả về số duy nhất đó.

Code: KTLT_ARRAY/src/singleNumber.cpp

```
1 int singleNumber(int nums[], int size);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/singleNumberXX.cpp

Gợi ý để giải quyết bài toán

1. **Phép XOR:** Sử dụng phép XOR để triệt tiêu các số trùng lặp, kết quả cuối cùng là số duy nhất.
2. **Bộ Nhớ Phụ (Mảng Đếm):** Đếm số lần xuất hiện của mỗi số và trả về số có đếm bằng 1.
3. **Sắp Xếp và Kiểm Tra Liên Tiếp:** Sắp xếp mảng và tìm số không trùng với số liền kề.

Phương Pháp	Time	Space	Ưu Điểm	Nhược Điểm
1. Phép XOR	O(n)	O(1)	Tối ưu về thời gian và không gian, dễ triển khai.	Ít trực quan hơn, khó hiểu cho người mới.
2. Bộ Nhớ Phụ (Hash Map/Mảng Đếm)	O(n)	O(n)	Đơn giản và dễ hiểu, dễ triển khai.	Sử dụng thêm bộ nhớ, không tối ưu về không gian.
3. Sắp Xếp và Kiểm Tra Liên Tiếp	O(nlogn)	O(1|n)	Dễ hiểu và dễ triển khai.	Độ phức tạp thời gian cao hơn, cần thay đổi mảng gốc.

Bảng 2: *So Sánh 3 Phương Pháp Đầu Tiên Giải Bài Toán* `singleNumber`

3 Hàm findSecondSmallest

Bài Toán: Bài toán findSecondSmallest có nhiệm vụ tìm và trả về giá trị nhỏ thứ hai trong một mảng các số nguyên. Nếu không tồn tại giá trị nhỏ thứ hai (ví dụ: mảng có kích thước nhỏ hơn 2 hoặc tất cả các phần tử trong mảng đều bằng nhau), hàm sẽ trả về một giá trị đặc biệt để biểu thị kết quả không hợp lệ (ví dụ: -1).

Code: KTLT_ARRAY/src/findSecondSmallest.cpp

```
1 int findSecondSmallest(int arr[], int size);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/findSecondSmallestXX.cpp

Gợi ý để giải quyết bài toán

1. **Sử dụng hai biến:** Duyệt mảng một lần, giữ giá trị nhỏ nhất và nhỏ thứ hai, cập nhật khi cần.
2. **Sắp xếp mảng:** Sắp xếp mảng tăng dần và tìm phần tử đầu tiên lớn hơn phần tử nhỏ nhất.
3. **Duyệt hai lần:** Lần đầu tìm giá trị nhỏ nhất, lần hai tìm giá trị nhỏ nhất trong các phần tử lớn hơn giá trị nhỏ nhất.

Phương Pháp	Time	Space	Ưu Điểm	Nhược Điểm
1. Sử dụng hai biến	O(n)	O(1)	Nhanh, hiệu quả, đơn giản	Có thể khó hiểu với người mới
2. Sắp xếp mảng	O(n log n)	O(1) hoặc O(n)	Dễ hiểu, tận dụng thư viện có sẵn	Không tối ưu về thời gian.
3. Duyệt hai lần	O(n)	O(1)	Đơn giản, dễ triển khai	Không tối ưu với số lượng dữ liệu lớn

Bảng 3: *So Sánh 3 Phương Pháp Đầu Tiên Giải Bài Toán* `findSecondSmallest`

4 Hàm maxSubArray

Bài Toán: Bài toán `maxSubArray` có nhiệm vụ tìm và trả về tổng lớn nhất của một dãy con liên tiếp trong một mảng các số nguyên. Nếu mảng rỗng hoặc không tồn tại dãy con hợp lệ, hàm trả về giá trị đặc biệt (ví dụ: 0).

Code: KTLT_ARRAY/src/maxSubArray.cpp

```
1 int maxSubArray(int arr[], int size);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/maxSubArrayXX.cpp

Gợi ý để giải quyết bài toán

1. **Sử dụng thuật toán Kadane:** Duyệt mảng một lần, giữ tổng lớn nhất tạm thời và tổng tối đa đã tìm được.
2. **Duyệt hai vòng lặp:** Thử tất cả các dãy con liên tiếp và tính tổng để tìm tổng lớn nhất.
3. **Chia để trị:** Chia mảng thành hai nửa, tìm tổng lớn nhất trong từng nửa và tổng lớn nhất qua phần giao giữa hai nửa.

Phương Pháp	Time	Space	Ưu Điểm	Nhược Điểm
1. Thuật toán Kadane	O(n)	O(1)	Nhanh, hiệu quả với thời gian tối ưu	Cần hiểu thuật toán Kadane
2. Duyệt hai vòng lặp	$O(n^2)$	O(1)	Dễ hiểu, dễ triển khai	Không phù hợp với mảng lớn
3. Chia để trị	O(nlog n)	O(log n)	Hiệu quả, phù hợp với các bài toán phức tạp	Khó triển khai hơn

Bảng 4: *So Sánh 3 Phương Pháp Giải Bài Toán `maxSubArray`*

5 Hàm isMountainArray

Bài Toán: Bài toán `isMountainArray` có nhiệm vụ kiểm tra xem một mảng số nguyên có phải là mảng "núi" hay không. Mảng "núi" là mảng thỏa mãn các điều kiện sau:

- Mảng có ít nhất 3 phần tử.
- Có một đỉnh (peak) sao cho:
  - Các phần tử trước đỉnh tăng dần.
  - Các phần tử sau đỉnh giảm dần.

Nếu thỏa mãn, hàm trả về `True`, ngược lại trả về `False`.

Code: KTLT_ARRAY/src/isMountainArray.cpp

```
1 bool isMountainArray(int arr[], int size);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/isMountainArrayXX.cpp

Gợi ý để giải quyết bài toán

1. **Duyệt một lần:**

   Duyệt mảng từ trái sang phải:

   - Xác định phần tăng dần để tìm đỉnh.
   - Tiếp tục kiểm tra phần giảm dần sau đỉnh.
   - Kiểm tra điều kiện đỉnh không nằm ở đầu hoặc cuối mảng.

2. **Duyệt hai lần:**
   - Lần 1: Tìm chỉ số của đỉnh bằng cách tìm phần tử lớn nhất trong mảng.
   - Lần 2: Kiểm tra tính chất tăng dần trước đỉnh và giảm dần sau đỉnh.

3. **Sử dụng mảng phụ:**
   - Tạo hai mảng phụ để lưu trữ thông tin:
     - Mảng `up[i]`: Đánh dấu phần tăng dần.
     - Mảng `down[i]`: Đánh dấu phần giảm dần.
   - Một mảng là "núi" nếu có ít nhất một vị trí `i` mà `up[i] = true` và `down[i] = true`.

Phương Pháp	Time	Space	Ưu Điểm	Nhược Điểm
1. Duyệt một lần	O(n)	O(1)	Nhanh, hiệu quả và dễ triển khai	Không phát hiện được nhiều đỉnh
2. Duyệt hai lần	O(n)	O(1)	Đảm bảo kiểm tra toàn diện	Chậm hơn phương pháp một lần duyệt
3. Sử dụng thuật toán mảng phụ	O(n)	O(n)	Dễ triển khai với mảng phụ	Tốn thêm không gian bộ nhớ

Bảng 5: *So Sánh 3 Phương Pháp Giải Bài Toán `isMountainArray`*

6 Hàm findMaxSumSubarray

Bài Toán: Bài toán `findMaxSumSubarray` có nhiệm vụ tìm và trả về tổng lớn nhất của một dãy con liên tiếp có độ dài đúng bằng `k` trong một mảng các số nguyên. Nếu không tồn tại dãy con có độ dài `k` hoặc mảng rỗng, hàm trả về giá trị đặc biệt (ví dụ: 0).

Code: KTLT_ARRAY/src/findMaxSumSubarray.cpp

```
1 int findMaxSumSubarray(int arr[], int size, int k);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/findMaxSumSubarrayXX.cpp

Gợi ý để giải quyết bài toán

1. **Sử dụng cửa sổ trượt (Sliding Window):**
   - Tính tổng ban đầu của `k` phần tử đầu tiên.
   - Duyệt qua mảng, tại mỗi bước:
     - Cập nhật tổng bằng cách trừ phần tử bị loại và thêm phần tử mới.
     - Cập nhật `maxSum` nếu tổng mới lớn hơn tổng hiện tại.
2. **Duyệt hai vòng lặp:**
   - Thử tất cả các dãy con liên tiếp có độ dài `k` bằng cách duyệt hai vòng lặp lồng nhau.
   - Tính tổng từng dãy con và so sánh để tìm tổng lớn nhất.
3. **Chia để trị:**
   - Chia mảng thành hai nửa:
     - Tìm tổng lớn nhất của dãy con có độ dài `k` trong nửa trái.
     - Tìm tổng lớn nhất của dãy con có độ dài `k` trong nửa phải.
     - Tìm tổng lớn nhất của dãy con có độ dài `k` đi qua phần giao giữa hai nửa.
   - Kết quả là giá trị lớn nhất trong ba tổng trên.

Phương Pháp	Time	Space	Ưu Điểm	Nhược Điểm
1. Cửa sổ trượt	O(n)	O(1)	Nhanh, hiệu quả, dễ triển khai	Chỉ áp dụng cho độ dài `k` cố định
2. Duyệt hai vòng lặp	$O(n^2)$	O(1)	Dễ hiểu, không cần thuật toán phức tạp	Chậm với mảng lớn
3. Chia để trị	O(nlog n)	O(log n)	Hiệu quả với bài toán phức tạp	Khó triển khai hơn

Bảng 6: *So Sánh 3 Phương Pháp Giải Bài Toán `findMaxSumSubarray` với Độ Dài `k`*

7 Hàm maxAreaOfIsland

Bài Toán: Bài toán `maxAreaOfIsland` có nhiệm vụ tìm và trả về diện tích lớn nhất của một "hòn đảo" trong ma trận nhị phân 2D.

Input: Ma trận grid

$$\text{grid} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Mô tả:

- Các ô giá trị `1` đại diện cho "đất".
- Các ô giá trị `0` đại diện cho "nước".
- Hòn đảo lớn nhất bao gồm các ô đất liền kề (ngang hoặc dọc).

Output: Diện tích hòn đảo lớn nhất là **5**.

Vùng hòn đảo lớn nhất (được đánh dấu):

$$\text{grid} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Code: KTLT_ARRAY/src/maxAreaOfIsland.cpp

```
// Hàm đệ quy tính diện tích hòn đảo
int dfs(int grid[][MAX], int i, int j, int n, int m);
int maxAreaOfIsland(int grid[][MAX], int n, int m);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/maxAreaOfIslandXX.cpp

Ý tưởng giải quyết bằng đệ quy (DFS):

1. **Xác định bài toán con:**
   - Khi gặp một ô `1` trong ma trận, ta sẽ khởi động một hàm đệ quy để tính diện tích hòn đảo bắt đầu từ ô đó.
   - Trong mỗi lời gọi đệ quy:
     - Nếu ô nằm ngoài giới hạn của ma trận hoặc có giá trị khác `1`, kết thúc đệ quy.
     - Nếu ô có giá trị `1`, tính diện tích hiện tại bằng cách cộng thêm 1.
     - Đệ quy mở rộng sang các ô liền kề (trái, phải, trên, dưới).
2. **Quy trình chính:**
   (a) Duyệt qua từng ô trong ma trận.
   (b) Nếu gặp ô `1`, gọi hàm đệ quy để tính diện tích hòn đảo bắt đầu từ ô đó.
   (c) Đánh dấu các ô đã thăm bằng cách thay đổi giá trị ô đó thành `0` để tránh tính lại.
   (d) Cập nhật diện tích lớn nhất tìm được sau mỗi lần gọi đệ quy.
3. **Kết thúc:**
   - Sau khi duyệt toàn bộ ma trận, kết quả là diện tích lớn nhất của một hòn đảo.
   - Nếu không có ô `1` nào trong ma trận, trả về `0`.

8 Hàm numberOfClosedIslands

Bài Toán: Bài toán `numberOfClosedIslands` có nhiệm vụ tìm và trả về số lượng "hòn đảo khép kín" trong một ma trận nhị phân 2D.

Định nghĩa hòn đảo khép kín: Một "hòn đảo khép kín" là một tập hợp các ô giá trị `1` (đất) được bao quanh hoàn toàn bởi các ô giá trị `0` (nước), bao gồm cả bốn cạnh của ma trận.

Input: Ma trận grid

$$\text{grid} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Mô tả:

- Các ô giá trị `1` đại diện cho "đất".
- Các ô giá trị `0` đại diện cho "nước".
- Một hòn đảo khép kín không tiếp xúc với rìa ma trận.

Output: Số lượng hòn đảo khép kín trong ma trận là **1**.

Vùng hòn đảo khép kín (được đánh dấu):

$$\text{grid\_marked} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Code: KTLT_ARRAY/src/numberOfClosedIslands.cpp

```
// Hàm đệ quy kiểm tra hòn đảo khép kín
bool dfs(int grid[][MAX], int i, int j, int n, int m);
int numberOfClosedIslands(int grid[][MAX], int n, int m);
```

Test Case: KTLT_ARRAY/test/unit_test_KTLT_array/test_case/numberOfClosedIslandsXX.cpp

Ý tưởng giải quyết bằng đệ quy (DFS)

1. **Xác định bài toán con:**
   - Khi gặp một ô `1` trong ma trận, ta sẽ khởi động một hàm đệ quy để kiểm tra xem hòn đảo bắt đầu từ ô đó có khép kín hay không.
   - Trong mỗi lời gọi đệ quy:
     - Nếu ô nằm ngoài giới hạn của ma trận, hòn đảo không khép kín (vì nó chạm rìa).
     - Nếu ô có giá trị `1`, đánh dấu là đã thăm bằng cách thay đổi giá trị thành `0`.
     - Tiếp tục mở rộng kiểm tra sang các ô liền kề (trái, phải, trên, dưới).
2. **Quy trình chính:**
   (a) Duyệt qua từng ô trong ma trận.
   (b) Khi gặp ô `1`, gọi hàm đệ quy để kiểm tra xem hòn đảo bắt đầu từ ô đó có khép kín hay không.
   (c) Nếu hòn đảo khép kín, tăng số lượng hòn đảo khép kín lên 1.
3. **Kết thúc:**
   - Sau khi duyệt toàn bộ ma trận, kết quả là số lượng hòn đảo khép kín.
   - Nếu không có hòn đảo khép kín nào, trả về `0`.