HƯỚNG DẪN GIẢI BÀI TẬP TWO POINTER

BTTP04: Tổng 2 phần tử

Yêu cầu: Cho một mảng số nguyên bắt đầu từ 1 và được sắp xếp theo thứ tự không giảm. Hãy tìm hai phần tử khác nhau trong mảng có tổng bằng một giá trị cho trước.

Dữ liệu:

- Dòng đầu chứa hai số nguyên n và k với n là số phần tử của mảng ($0 <= n <= 10^6$). k là giá trị số tìm.
- Dòng tiếp theo chứa n số nguyên cách nhau khoảng trắng là giá trị các phần tử trong mảng A.

Kết quả: Ghi 2 số i, j là vị trí mà tổng 2 phần tử trong mảng có giá trị bằng k. Trường hợp có nhiều kết quả thì lấy vị trí i ở bên trái nhất và vị trí j ở bên phải nhất. Nếu không tìm được kết quả xuất ra -1

Ví dụ:

Dữ liệu	Kết quả
3 6 2 3 4	1 3

Hướng dẫn giải:

Dạng bài: two pointer dạng left - right

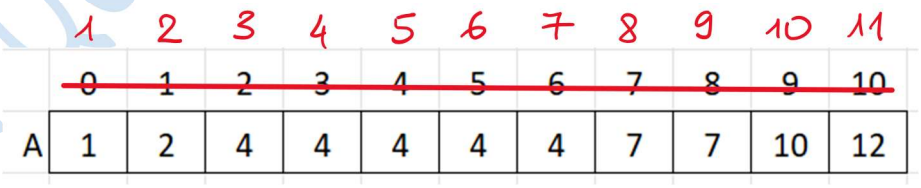
Cấu trúc dữ liệu:

```
const int nmax = 1e6;
int m[nmax + 1];
```

- Mảng số nguyên M lưu dữ liệu đầu vào

Giải thuật:

- Đề bài yêu cầu tìm 2 phần tử bất kỳ trong mảng có tổng bằng K, trường hợp có nhiều kết quả thì lấy kết quả i nhỏ nhất và j lớn nhất.
- Vì mảng đã cho có thứ tự tăng dần nên ta tạo 2 con trỏ L = 1 và R = N
- Trong khi M[L] + M[R] khác K và L còn nhỏ hơn R thì
  - Nếu M[L] + M[R] < K thì tăng L ngược lại giảm R
  - Nếu M[L] + M[R] == K thì xuất L, R và thoát chương trình
- Khi vòng lặp kết thúc mà L >= R thì xuất -1



Độ phức tạp O(N)

BTTP05: Tập hợp đoạn con

Yêu cầu:

Cho trước một dãy số A gồm n số nguyên là các bit 0 và 1 cùng với một số nguyên dương k.

Yêu cầu: Hãy tìm một tập gồm nhiều nhất các ***đoạn con*** của dãy A sao cho kích thước của các đoạn con đó đều đôi một khác nhau, và tổng của các đoạn con đều bằng k?

Dữ liệu:

- Dòng đầu cho 2 số nguyên dương n và k ($1 <= k <= n <= 10^7$)
- Dòng thứ 2 chứa một dãy gồm n số 0, 1 cách nhau khoảng trắng

Kết quả: Ghi ra kích thước lớn nhất của tập hợp các đoạn con tìm được. Nếu không tìm được ghi ra 0

Ví dụ:

Dữ liệu	Kết quả
4 2 0 1 1 0	3

Hướng dẫn giải:

Dạng bài: two pointer dạng slow and fast

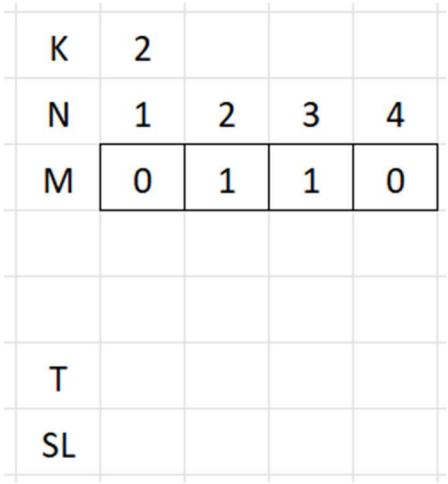
Cấu trúc dữ liệu:

```
const int nmax = 1e7+5;
int m[nmax], n, k;
unordered_set <int> kq;
```

- Mảng số nguyên M lưu dữ liệu đầu vào
- Set không sắp xếp thứ tự KQ để lưu số lượng mảng con có kích thước khác nhau thõa yêu cầu đề bài.

Giải thuật:

- Nhận xét: thuật ngữ "đôi một khác nhau" có nghĩa là tất cả các phần tử trong tập hợp đang xét đều phải khác nhau. Vậy đề bài yêu cầu tìm tất cả các mảng con có kích thước khác nhau và có tổng bằng K.
- Giải thích ví dụ: Mảng M: [0, 1, 1, 0] và K = 2 ta có các mảng con sau thõa điều kiện [1, 1] [0, 1, 1] [0, 1, 1, 0]
- Các bước thực hiện
    - Tạo biến con trỏ chậm S = 1, biến con trỏ nhanh F = 1 (biến đếm vòng lặp) biến lưu tổng giá trị đoạn con SUM = 0, biến số lượng các phần tử của đoạn con SL = 0
    - Duyệt F từ 1 đến N
        - SL++
        - SUM += M[F]
        - Nếu SUM < K thì tiếp tục tăng F
        - Nếu SUM > K thì
            - SUM -= M[S]
            - S++
            - SL—
        - Nếu SUM = K
            - Thêm giá trị SL vào KQ
            - Tạo biến tạm T = S
            - Trong khi M[T] = 0 thì
                - Thêm giá trị F – T vào KQ
                - T++



Độ phức tạp O(N)

BTTP06: Tổng 3 phần tử

Yêu cầu: Cho mảng số nguyên có n phần tử, tìm tất cả các tổng có 3 phần tử ai + aj + ak = 0 với (i != j, i != k và j != k). Nếu có nhiều bộ ba phần tử giống nhau thì chỉ lấy một.

Dữ liệu: Cho các phần tủ số nguyên của mảng, mỗi phần tử cách nhau một khoảng trắng. Độ dài tối đa của mảng là 10000 phần tử

Kết quả:

- Dòng đầu ghi tổng số bộ ba khác nhau có tổng bằng không tìm được.
- Các dòng tiếp theo, mỗi dòng ghi 3 số cách nhau khoảng trắng là một bộ ba có tổng bằng không

Ví dụ:

Dữ liệu	Kết quả
-1 0 1 2 -1 -4	2 -1 -1 2 -1 0 1

Hướng dẫn giải:

Dạng bài: two pointer dạng slow and fast kết hợp left right (tree pointer)

Cấu trúc dữ liệu:

```
const int nmax = 1e4;
int m[nmax], n = 0;
set<vector<int>> kq;
```

- Mảng số nguyên M lưu dữ liệu đầu vào
- Set kiểu vector số nguyên KQ lưu các bộ 3 phần tử khác nhau có tổng bằng 0

Giải thuật:

- Nhận xét:
  - Do đề bài yêu cầu nếu có nhiều bộ 3 phần tử giống nhau thì chỉ lấy 1 nên ta sử dụng cấu trúc dữ liệu set kết hợp với vector.
  - Để 3 phần tử khác nhau cộng lại có tổng bằng không thì bắt buộc trong 3 phần tử đó phải có phần tử số âm. Để tăng tốc độ bài toán ta nên sắp xếp lại mảng có thứ tự tăng dần.
- Các bước thực hiện:
  - Sắp xếp mảng M tăng dần
  - Tạo biến con trỏ chậm S = 0 (biến đếm vòng lặp), Tạo biến con trỏ nhanh đồng thời là biến con trỏ bên trái FL = S+1, tạo biến con trỏ bên phải R = N-1
  - Duyệt S từ phần tử đầu tiên đến cuối mảng
    - Nếu M[S] = 0 thì thoát và xuất kết quả
    - Nếu S > 0 và M[S] = M[S-1] thì tăng S
    - Ngược lại
      - FL = S+1
      - R = N-1
      - Trong khi FL < R thì
        - SUM = M[S] + M[FL] + M[R]
        - Nếu tổng của SUM = 0 thì
          - Thêm vào KQ
          - Tăng FL
          - Giảm R

- Nếu SUM < 0 thì tăng FL
- Nếu SUM > 0 thì giảm R

- Duyệt set KQ để xuất kết quả

	0	1	2	3	4	5
M	-1	0	1	2	-1	-4

M	-4	-1	-1	0	1	2

Độ phức tạp O(logN + N)

BTTP7: Đọc sách

Yêu cầu: Trong thư viện trường có N quyển sách. Cuốn sách thứ i mất thời gian Ai để đọc. Mỗi lần đến thư viện trường Minh có T thời gian. Minh muốn đọc một dãy các quyển sách liên tiếp nhau dài nhất có thể. Hỏi Minh có thể đọc tối đa bao nhiêu quyển sách.

Dữ liệu:

- Dòng đầu cho 2 số nguyên dương N ($0 < N <= 10^6$), T ($0 < T < 10^9$) là số quyển sách có trong thư viện và thời gian của Minh
- Dòng tiếp theo ghi N số nguyên là thời gian để đọc của các quyển sách, quyển sách thứ i sẽ có thời gian đọc là Ai ($0 < Ai < 10^9$)

Kết quả: ghi một số nguyên đó là tổng số sách liên tiếp lớn nhất mà Minh có thể đọc được

Ví dụ:

Dữ liệu	Kết quả
4 5 3 1 2 1	3

Hướng dẫn giải:

Dạng bài: two pointer dạng slow and fast kết hợp sliding windows

Cấu trúc dữ liệu:

```
const int nm = 1e6+5;
int a[nm], n, t;
```

- Mảng số nguyên A lưu dữ liệu đầu vào

Giải thuật:

- Nhận xét:
  - Đây là dạng bài sliding windows với kích thước của số thay đổi.
  - Ta sẽ linh động điều chỉnh kích thước của sổ để luôn chứa nhiều nhất các quyển sách liên tiếp mà tổng thời gian đọc không quá T.
- Các bước thực hiện:
  - Khởi tạo:
    - L = 1; con trỏ chậm bắt đầu từ quyển sách đầu tiên
    - R = 1; con trỏ nhanh bắt đầu từ quyển sách đầu tiên
    - current_sum = 0; tổng thời gian đọc các quyển sách trong phạm vi cửa sổ từ [S, R-1]
    - max_length = 0; lưu số lượng sách liên tiếp lớp nhất có thể đọc
  - Duyệt R từ 1 đến N:
    - Mở rộng cửa sổ: Đưa cuốn sách ở vị trí R vào cửa sổ, current_sum += A[R]
    - Kiểm tra và thu hẹp cửa sổ: Trong khi current_sum > T có nghĩa là tổng thời gian đọc sách đã quá thời gian cho phép, ta phải giảm thời gian đọc sách bằng cách thu hẹp cửa sổ ở phía bên trái bằng cách bỏ cuốn sách tại vị trí L ra
      - current_sum -= A[L]
      - S++
    - Lặp lại việc kiểm tra thu hẹp cửa sổ cho đến khi cửa sổ hợp lệ (current_sum <= T)
    - Cập nhật max_length
  - Xuất kết quả:

N	16															
T	11															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	0	2	4	1	1	1	9	8	1	2	4	1	4	1	9	8	3

Độ phức tạp O(N)

BTTP8: Vòng đu quay

Yêu cầu: Trong một công viên giải trí, trò chơi vòng đu quay được rất nhiều trẻ em tham gia. Mỗi cabin trên đu quay có thể chứa một hoặc hai đứa trẻ. Tổng cân nặng trong một cabin không được vượt quá X. Có một hàng N đứa trẻ xếp hàng để tham gia trò chơi, trước khi chơi, mỗi đứa trẻ đều được cân và ghi nhận lại cân nặng. Hãy giúp người quản lý trò chơi tìm ra số cabin ít nhất được dùng.

Dữ liệu:

- Dòng đầu cho 2 số nguyên dương N ($0 < N <= 10^6$), X ($0 < X < 10^4$) là số cabin trên vòng đu quay và tải trọng của một cabin
- Dòng tiếp theo ghi N số nguyên là cân nặng của từng đứa trẻ, đứa trẻ thứ i sẽ có cân nặng là Ai ($0 < Ai < 100$)

Kết quả: ghi ra một số nguyên dương là số cabin ít nhất được sử dụng

Ví dụ:

Dữ liệu	Kết quả
5 50 45 35 24 23 37	4

Hướng dẫn giải:

Dạng bài: two pointer dạng left – right kết hợp tư tưởng tham lam

Cấu trúc dữ liệu:

```
const int nm = 1e6+5;
int a[nm], n, t;
```

- Mảng số nguyên A lưu dữ liệu đầu vào

Giải thuật:

- Phân tích đề bài:
  - Mục tiêu: "số lượng cabin tối thiểu". Điều này thường gợi ý đến một chiến lược tối ưu, có thể là Tham lam.
  - Làm thế nào để tối thiểu hóa số cabin? Bằng cách tối đa hóa số cặp trẻ em được ghép chung một cabin.
  - Chiến lược Tham lam (Greedy): Để có cơ hội ghép cặp tốt nhất cho những đứa trẻ nặng nhất (vốn rất khó ghép), ta nên thử ghép chúng với những đứa trẻ nhẹ nhất có thể. Nếu đứa trẻ nặng nhất còn lại có thể đi cùng đứa trẻ nhẹ nhất còn lại, ta nên thực hiện việc ghép cặp này. Tại sao? Vì nếu đứa trẻ nhẹ nhất này không thể đi cùng đứa nặng nhất, thì chắc chắn nó cũng không thể đi cùng bất kỳ đứa nào nặng hơn. Việc ghép cặp này giải quyết được cả hai "trường hợp khó" cùng lúc.
- Hướng dẫn giải: Chiến lược tham lam trên dẫn chúng ta đến một thuật toán sử dụng Two Pointers ngược chiều.
  1. Sắp xếp: Đầu tiên, sắp xếp mảng cân nặng của trẻ em theo thứ tự tăng dần.
  2. Hai Con Trỏ: Khởi tạo left = 0 (đứa trẻ nhẹ nhất) và right = n-1 (đứa trẻ nặng nhất). Khởi tạo DEM = 0.
  3. Vòng lặp: Trong khi left <= right:
     a. Ta chắc chắn sẽ dùng một cabin mới cho đứa trẻ nặng nhất (tại right). Vì vậy, ta tăng DEM lên 1.
     b. Kiểm tra xem đứa trẻ nhẹ nhất (tại left) có thể đi cùng không: if (left < right && weights[left] + weights[right] <= x).
     c. Nếu có thể, ta ghép cặp thành công. Cả hai đứa trẻ đều đã có chỗ. Ta di chuyển cả hai con trỏ: left++ và right--.
     d. Nếu không thể, đứa trẻ nặng nhất phải đi một mình. Ta chỉ di chuyển con trỏ right--. Đứa trẻ nhẹ nhất tại left sẽ được xét ở lượt sau.
  4. Khi left > right, tất cả trẻ em đã được xếp vào cabin. DEM là kết quả.

Độ phức tạp O(N)