Hướng dẫn giải Code Challenge #6

Big-O & VNG - 16/11/2022

BÀI A: Jimmy and Lucky Money

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Nhận xét 1:

Thay vì ta bỏ các tờ tiền trong 1 bao lì xì vào ví ở vị trí bất kì rồi sắp xếp lại số đoạn bị sai, ta có thể *tham lam* để bao lì vào vị trí hợp lí ở lần đầu tiên sau đó mới sắp xếp lại các đoạn bị sai vị trí (sẽ giảm đi được 1 thao tác)

Ví dụ:

- Ví: $[1, 2, 2, 5]$
- Bao lì xì: $[2, 5, 10]$
- Ta sẽ *tham lam* bỏ bao lì xì vào vị trí 2 trong ví (đánh số từ 0) sẽ tiết kiệm được 1 thao tác sắp xếp lại số 2 trong bao lì xì lúc sau.

Nhận xét 2:

Thay vì thao tác sắp xếp lại vị trí của các đoạn bị sai sau khi bỏ bao lì xì vào ví (theo *nhận xét 1*). Ta có thể chia bao lì xì ra thành các đoạn thích hợp dựa theo tư tưởng *tham lam* sau đó mới bỏ chúng vào ví, số thao tác cũng sẽ như nhau cho 2 cách trên.

Ví dụ:

- Ví: $[1, 6, 9, 11, 12, 14, 15]$
- Bao lì xì: $[2, 2, 3, 4, 4, 11, 14]$
- Nếu làm theo cách đề bài và kết hợp *nhận xét 1*:
  - Bước 1: bỏ bao lì xì vào trong ví (số thao tác = 1)
    - Ví: $[1, 2, 2, 3, 4, 4, 11, 14, 6, 9, 11, 12, 14, 15]$
  - Bước 2: Sắp xếp các đoạn sai về lại đúng vị trí (số thao tác = 2)
    - Thay đổi đoạn $[11]$ bắt đầu từ vị trí 6. Ví: $[1, 2, 2, 3, 4, 4, 14, 6, 9, 11, 11, 12, 14, 15]$
    - Thay đổi đoạn $[14]$ bắt đầu từ vị trí 6. Ví: $[1, 2, 2, 3, 4, 4, 6, 9, 11, 11, 12, 14, 14, 15]$
  - Bước 3: Xuất ra kết quả = 3
- Tuy nhiên nếu kết hợp với *nhận xét thứ 2*, ta có thể chia bao lì xì ra thành 3 đoạn $[2, 2, 3, 4, 4]$, $[11]$, và $[14]$ sau đó bỏ chúng vào ví, cũng tốn 3 thao tác

Ý tưởng:

Làm thế nào để chia bao lì xì ra thành các đoạn sao cho số đoạn là ít nhất.

Tạo 1 set để lưu giữ các mệnh giá tờ tiền đã xuất hiện. Ta sẽ dùng set này để tìm những giá trị lớn hơn gần nhất so với tờ tiền đang xét bằng hàm $upper_bound$

Tạo 1 biến $bound$ lưu giá trị lớn nhất có thể có của đoạn đang xét. Những tờ tiền có mệnh giá $< bound$ sẽ được thêm vào đoạn (tư tưởng tham lam). Nếu mệnh giá tờ tiền đang xét $ x > bound$, ta cập nhật lại $bound = set.upper\_bound(x)$ để chuẩn bị cho đoạn tiếp theo bắt đầu từ x.

Các bước giải:

- $INF = 10^9 + 1$
- Khởi tạo $set = \{INF\}$. Ta cần khởi tạo set có giá trị là INF để khi không tìm thấy $upper\_bound(x)$ thì ta sẽ trả về INF. Tức số x là số lớn nhất.

Lần lượt duyệt qua từng bao lì xì:

- Khởi tạo $ans = 1$ (đã bỏ đoạn đầu tiên vào trong ví).
- Khởi tạo $bound = upper\_bound(0)$ (vì mệnh giá tờ tiền nằm trong khoảng $[1, 10^9]$).
- Khởi tạo $first\_range\_empty = true$. Đây là biến để check xem đã chọn được vị trí cho range đầu tiên hay chưa.
- Lần lượt duyệt qua từng tờ tiền trong bao lì xì:
  - Gọi x là mệnh giá tờ tiền đang xét
  - Nếu $x == bound$ và $first\_range\_empty == True$: (đã chọn được vị trí cho đoạn đầu $ans = 1$):
    - Cập nhật $bound = upper\_bound(x)$ để tìm những tờ tiền thuộc về đoạn đầu tiên.
  - Nếu $x > bound$: (x là bắt đầu cho đoạn thứ tiếp theo)
    - Cập nhật: ans += $!first\_range\_empty$ (Nếu chưa tìm được vị trí cho đoạn đầu tiên thì sẽ tiếp tục tìm mà không +1 cho ans).
    - Cập nhật $bound = upper\_bound(x)$ để tìm những tờ tiền cho đoạn tiếp theo.
  - Cập nhật $first\_range\_empty == False$
  - Thêm x vào trong set
- Ví dụ:
  - Ví: $[1, 2, 2, 5]$
  - Bao lì xì: $[2, 5, 10]$
  - $first\_range\_empty = true$
  - $set = \{1, 2, 5, INF\}$
  - Duyệt qua từng tờ tiền trong bao:
    - $bound = upper\_bound(0) = 1$
    - $ans = 1$
    - $x = 2 > bound$:
      - $bound = upper\_bound(2) = 5$
      - $ans \mathrel{+}= 0$
      - $first\_range\_empty == False$
      - Thêm 2 vào trong set
    - $x = 5 \leq bound$:
      - $first\_range\_empty == False$
      - Thêm 5 vào trong set
    - $x = 10 > bound$:
      - $bound = upper\_bound(10) = INF$
      - $ans \mathrel{+}= 1$
      - $first\_range\_empty == False$
      - Thêm 10 vào trong set
  - In kết quả: $ans = 2$

Lưu ý: Khi sử dụng ngôn ngữ Python, ta không dùng thể dùng set vì set trong Python không lưu các số tăng dần dẫn đến không tìm được $upper\_bound$. Ta có thể mô phỏng việc sử dụng set bằng cách sử dụng AVL Tree để cài đặt. Chi tiết xem tại đáp án.

Time complexity: $O(N \times logN)$ (N là tổng số lượng tờ tiền).

Mã nguồn tham khảo

- C++: https://ideone.com/c901cF

BÀI B: Kiki - Tom and Jerry

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Các bước giải:

Bước 1: Đọc vào chuỗi s, n và k

Bước 2: Khởi tạo các biến:

- $cntCheese$ dùng để đếm số lượng miếng phô mai liền kề nhau.
- $cntObstacle$ dùng để đếm số lượng chướng ngại vật liền kề nhau.
- $cntStop$ dùng để đếm số lần Jerry dừng lại để tiêu hóa phô mai.
- $belt$ dùng để đếm số lượng miếng phô mai trong bụng Jerry ở thời điểm đang xét.

Bước 3: Duyệt lần lượt từng kí tự trong chuỗi s

1. Nếu kí tự là '*':

- Tăng biến $cntCheese$ lên 1 đơn vị và so sánh với k. Nếu $cntChesse > k$, chuyển sang bước 3.
- Tăng biến $belt$ lên 1 đơn vị và so sánh với n. Nếu $belt = n$, tăng biến $cntStop$ lên 1 đơn vị và khởi tạo lại $belt$ = 0. Lưu ý, nếu đây là kí tự cuối của chuỗi thì ta không cần tăng biến $cntStop$ nữa, vì lúc này Jerry đã bị sập bẫy.
- Khởi tạo lại $cntObstacle$ = 0

2. Nếu kí tự là '#'

- Tăng biến $cntObstacle$ lên 1 đơn vị và so sánh với k. Nếu $cntObstacle > k$, chuyển sang bước 4.
- Vì thời gian đi qua 1 chướng ngại vật và thời gian tiêu hóa 1 miếng phô mai đều là 1 giây, nên khi vượt qua 1 chướng ngại vật, Jerry sẽ tiêu hóa 1 miếng phô mai trong bụng (nếu có). Vậy nên ta sẽ giảm $belt$ đi 1 đơn vị.
- Khởi tạo lại $cntCheese$ = 0

Bước 4: In kết quả

- Nếu $cntCheese > k$ hoặc $cntObstacle > k$ thì in "FAIL".
- Ngược lại, in thời gian để Jerry sập bẫy là độ dài của chuỗi $s + cntStop * n$

Độ phức tạp: $O(n)$.

Mã nguồn tham khảo

- C++: https://ideone.com/blhhpj

BÀI C: Dãy MEX

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Đầu tiên ta cần tìm MEX của dãy A ban đầu. Giả sử giá trị này là m. Lưu ý điều này có nghĩa là tất cả các giá trị 0, 1, 2, ..., m-1 đều xuất hiện trong A. Để tìm giá trị này, ta chỉ cần cho hết tất cả các giá trị của A vào một bảng băm, rồi sau đó duyệt hết các giá trị từ 0 đến N, giá trị đầu tiên không xuất hiện trong A chính là m. Do A chỉ có N phần tử, giá trị MEX lớn nhất của A chỉ có thể là N.

Việc loại bỏ bất kì một phần tử nào của A mà có giá trị lớn hơn m đều không làm thay đổi giá trị MEX này. Hãy nhớ lại định nghĩa về MEX là số nguyên không âm nhỏ nhất không xuất hiện trong tập đó.

Vậy chuyện gì sẽ xảy ra nếu ta loại bỏ một phần từ A[i] < m? Nếu A[i] là giá trị chỉ xuất hiện một lần trong A, việc loại bỏ A[i] sẽ làm cho số nguyên không âm nhỏ nhất không thuộc về A chính là A[i]. Nếu A[i] xuất hiện từ hai lần trở lên, giá trị MEX cũng sẽ không thay đổi.

Như vậy ta chỉ cần dùng một bảng băm để lưu số lần xuất hiện của mỗi giá trị trong A. Với mỗi phần tử A[i], nếu A[i] > m thì B[i] tương ứng sẽ có giá trị là m. Với A[i] < m, sẽ có hai trường hợp. Nếu A[i] chỉ xuất hiện 1 lần thì B[i] sẽ có giá trị là A[i] nếu không thì B[i] sẽ có giá trị là m.

Mã nguồn tham khảo

- C++: https://ideone.com/cH4Uru

BÀI D: Ghép nhóm UpRace

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Đầu tiên, cần phải đếm được có bao nhiêu người của mỗi nhóm. Chúng ta có thể dùng hashmap để lưu giá trị (key, value) là (nhóm số mấy, số người của nhóm). Duyệt mảng ban đầu để đếm, nếu số ak đó chưa từng xuất hiện thì lưu vào hashmap giá trị là $$(a_{k}, 1)$$. Còn nếu ngược lại thì chỉ cần tăng value của key $$a_{k}$$ đó thêm 1. Sau khi có được hashmap lưu số lần xuất hiện của mỗi số, chúng ta tìm các nhóm có số lượng người trùng nhau, và giảm dần số người cho đến khi số lượng người của mỗi nhóm là khác nhau. Để làm điều đó, ta duyệt mỗi phần tử trong hashmap:

- Với mỗi phần tử, nếu phần tử này có giá trị duy nhất trong hashmap, tương ứng với việc nhóm này có số thành viên không trùng lắp với nhóm khác, thì chúng ta sẽ xét phần tử tiếp theo
- Nếu phần tử này không là duy nhất, chúng ta bỏ đi 1 người, và lặp lại bước trên. Sau khi duyệt xong hashmap, số lượng người chúng ta bỏ đi là đáp án cần tìm.

Độ phức tạp: $$O(N^3)$$ với $$N$$ là số lượng phần tử.

Mã nguồn tham khảo

- C++: https://ideone.com/jGgbwx