Rời rạc hoá và ứng dụng

Rời rạc hoá và ứng dụng

Tác giả: Lê Hùng Sơn - Đại học FPT

Giới thiệu

Rời rạc hóa là một lĩnh vực lớn có đối tượng nghiên cứu là các tập hợp rời rạc trong khoa học máy tính. Ứng dụng của của phương pháp rất lớn và thường được sử dụng trong rất nhiều kỳ thi lớn. Trong phạm vi chuyên đề ta chỉ xét một số ví dụ để hiểu rõ thêm phương pháp này.

Khi giải thuật lập trình ta hay quen gọi phương pháp rời rạc hóa là **nén số**. Thật vậy, đúng như tên gọi, bản chất của phương pháp ta hiểu nôm na là đưa các *vùng dữ liệu lớn* về *vùng dữ liệu nhỏ* để dễ xử lý, sao cho vẫn thỏa mãn yêu cầu của bài toán đặt ra.

Kỹ thuật bổ trợ trong phương pháp này là *đánh lại số thứ tự* hay còn được gọi là nén số, được thực hiện như sau:

- Giả sử ta nén số một mảng A_i có n phần tử có giá trị thuộc khoảng $[-10^9, 10^9]$ về mảng nhỏ hơn có giá trị thuộc khoảng $[1, n]$ mà vẫn đảm bao được quan hệ lớn bé.

Ví dụ: a = {100, 100, 2000, 1500, 900000} → b = {1,1,3,2,4}

- *B1:* Dùng 2 mảng *song song* `val[i] = a[i], pos[i] = i` (`pos` để lưu vị trí đi kèm giá trị `a[i]`)
- *B2:* Sắp xếp lại theo tăng dần của `val[ ]` chú ý khi `swap(val[i],val[j])` nhớ `swap(pos[i],pos[j])` .
- *B3:* Tạo một biến `dem = 0, last = max` , duyệt các giá trị `val[i]` nếu `last` khác `val[i]` thì: `dem++, last = val[i];` ở mỗi bước ta cập nhật `b[pos[i]] = dem` .

Kết thúc quá trình trên, ta nhận được mảng `b[]` là nén từ mảng `a[]` với độ phức tạp thao tác nén này là $O(n * log(n))$.

Ví dụ 1: Dãy số (C11SEQ)

Đề bài

Cho n số nguyên $(n \leq 10^5)$ số nguyên $a_1, a_2, \ldots, a_n$ với $(||a_i|| \leq 10^9)$ và 2 số L, R $(L \leq R)$. Hãy đếm xem có bao nhiêu cặp (i, j) thỏa $L \leq a_i + a_{i+1} + \ldots + a_j \leq R$.

Input:

- Dòng đầu chứa 3 số n, L, R.
- Dòng 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$.

Output:

- In ra kết quả cần tìm.

Example:

```
C11SEQ.INP

4 2 4
1 2 3 4

C11SEQ.OUT
4
```

Hướng giải quyết:

- Hướng đơn giản nhất là duyệt mọi cặp đoạn (i, j) và kiểm tra xem tổng nó có thỏa không và ta tăng biến đếm lên. Tuy nhiên cách này mất chi phí thời gian $O(n^2)$ với $n \leq 10^5$ thì không được khả thi.
- Bây giờ ta thử gọi như sau: $S_i = a_1 + a_2 + \cdots + a_i$.
- Đoạn con (i, j) $(i \geq j)$ thỏa mãn điều kiện nếu $L \leq S_i - S_{j-1} \leq R$. Biến đổi tiếp ta được 2 điều kiện để thỏa là: $S_i - L \geq S_{j-1}$ và $S_i - R \leq S_{j-1}$.

Tiếp theo, ta có:

- Nhận xét 1: $S_i - L,\ S_i - R$ là 2 số cố định.
- Nhận xét 2: Quan hệ <= hay >= cho ta thấy: không cần quan tâm giá trị của các số mà chỉ cần đảm bảo quan hệ <= hay >= là được. Ví dụ: 1 < 5 ta có thể nén thành 1 < 2 chả ảnh hưởng kết quả bài toán.
- Nhận xét 3: Quá lắm chỉ có $3 * n$ phần tử cho tất cả các giá trị: $S_i - L, S_i - R, S_{j-1}$, với $n \leq 10^5$ thì đây là con số nhỏ.

Từ 3 nhận xét trên ta sẽ tìm cách đưa $S_i - L, S_i - R, S_{j-1}$ về các mảng nhỏ không quá $3 * n$ phần tử để dễ dàng quản lý:

- Ta lập một mảng mới có 3*n phần tử: n phần tử dạng S_i, n dạng $S_i - L$, n dạng $S_i - R$, nhớ lưu vị trí đi kèm.
- Bây giờ tiến hành sort mảng đó lại, và ta tiến hành đánh số lai mảng đó, gọi các mảng $p1_i, p2_i, p3_i$ là các giá trị sau khi đánh số lại của $S_i, S_i - L, S_i - R$.
- Ta tiến hành duyệt các vị trí i, dùng 1 cây Segment Tree hoặc Binary Indexed Tree để quản lý và đếm:
  - B1: cập nhất kết quả: tăng res thêm số lượng phần tử đoạn $[p3_i, p2_i]$ đã xuất hiện.
  - B2: thêm số lượng 1 phần tử $p1_i$ vào cây.
- Độ phức tạp: $O(3 * n * log(3 * n))$.

- Ngoài cách này ra, ta còn 1 cách dùng *Phương pháp chia để trị*, sẽ có trong các tài liệu sắp tới.

Code tham khảo (pascal):

```
// Code phần nén số:
// ở đây thay vì dùng 3 mảng p1[i], p2[i], p3[i] mình tận dụng luôn mảng a:
// * a[i] = p1[i], a[n + i] = p2[i], a[2*n + i] = p3[i]
procedure unzip;
var i,j,del:longint;
begin
    sort(1,3*n);
    A[3*n+1].val:=high(longint);
    i:=1;  del:=0;
    repeat
         inc(del);
         j:=i;
         while A[i].val=A[j].val do
         begin
             B[A[j].pos]:=del;
             inc(j);
         end;
         i:=j;
    until i=3*n+1;
end;

// Phần tính toán kết quả bằng Binary Indexed Tree
for i:=n downto 2 do
    begin
        update(B[i]);
        res:=res+get(B[i-1+2*n])-get(B[i-1+n]);
    end;
```

Ví dụ 2: Phân đoạn (VOI 2005 - Bảng A)

Cho dãy n số nguyên $a_1, a_2, \ldots, a_n$ và số $k\,(1 \leq n, k \leq 15000)\,(\|a_i\| \leq 30000)$ hãy tìm số m nhỏ nhất sao cho có thể chia dãy đã cho thành k phần, mỗi phần là 1 đoạn các phân tử liên tiếp, và phải đảm bảo tổng mỗi phần không quá m.

Input:

- Dòng đầu chứa số nguyên n và k.
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$.

Output:

- In ra số nguyên m.

Example:

```
QBSEGPAR.INP

9 4
1 1 1 3 2 2 1 3 1

QBSEGPAR.OUT

5
```

Hướng giải quyết

Nhận xét 1: Bài toán yêu cầu tìm m nhỏ nhất, theo kinh nghiệm thì khi bài toán bảo tìm giá trị nhỏ nhất hay lớn nhất nhưng không xác định được từ dữ liệu bài thì ta nên nghĩ đến *chặt nhị phân*. Vùng giá trị chặt có thể chọn từ $[-10^9, 10^9]$ là vừa hợp, cái này là tùy chọn, còn tối ưu nhất chỉ cần chặt trong khoảng $[-max(a_i) * n, max(a_i) * n]$.

- Tuy nhiên, ta chỉ dự đoán là chặt nhị phân nhưng chưa khẳng định là có đúng không, ta có nhận xét sau: _với m càng lớn thì việc chia thành k đoạn càng dễ _ → dùng chặt nhị phân là chính xác.

Nhận xét 2: Nếu ta có 1 giá trị m xác định, ta chia được ít nhất là a đoạn, chia nhiều nhất là b đoạn, nếu tồn tại k mà $a \leq k \leq b$ thì luôn có cách chia k đoạn thỏa mãn. Để xác định được a và b ta dùng phương pháp *Quy hoạch động*.

Chặt nhị phân không khó, ở đây khó là phương pháp _quy hoạch động _cho thỏa mãn thời gian. Công thức sơ khai như sau:

- $S_i = a_1 + a_2 + \cdots + a_i$.
- $fmax_i$ = số đoạn chia được nhiều nhất trong dãy $a_1, a_2, \ldots, a_i$.
- $fmin_i$ = số đoạn chia được ít nhất trong dãy $a_1, a_2, \ldots, a_i$.

Khởi tạo: `fmax[0] = 0, fmin[0] = 0, fmax[i] = -max (i != 0), fmin[i] = INF (i != 0)`.

Công thức:

- `fmax[i] = max(fmax[i], fmax[j] + 1)` với `j < i` và `S[i] - S[j] <= m`.
- `fmin[i] = min(fmin[i], fmin[j] + 1)` với `j < i` và `S[i] - S[j] <= m`.

Nhận thấy độ phức tạp đây là $O(n^2 * log(2 * 10^9))$ không thể đáp ứng được thời gian yêu cầu là 1s nhưng ở trường hợp quá bí ý tưởng đây không phải giải pháp tồi giúp lấy được một ít điểm lẻ.

Để nhanh được chỉ có cách là cải tiến sao cho tính mảng Quy hoạch động được nhanh, ở đây ta để ý quan hệ $S_i - S_j \leq m$ chỉ cần biến đổi thành $S_i - m \leq S_j \rightarrow$ giải pháp đã phần nào sáng sủa hơn và nếu tinh ý thì đây chỉ là bài toán 1 chiều, "một nửa" của **ví dụ 1** ở trên thôi → Bây giờ ta chỉ cần rời rạc hóa nó đi thay vì $3 * n$, ta có mảng $2 * n$ lưu các giá trị S_i và $S_i - m$, ta sẽ tính dựa vào 1 cây Binary Indexed Tree cho đơn giản thay vì đếm như bài trên, vấn đề ở đây chỉ là tìm max min, và update max, min.

- Độ phức tạp: $O(n * log(n) * log(2 * 10^9))$.