ĐÁP ÁN PREVNOI 2017, NGÀY 1

Bài 1. PHÁT QUÀ (Thầy Nguyễn Thanh Hùng)

Tính mảng $b[1 \ldots n-k+1]$, ở đây $b[i]$ tổng của k phần tử liên tiếp trong dãy A tính từ vị trí i:

$$b_i = a_i + a_{i+1} + \cdots + a_{i+k-1}$$

Điều này có thể thực hiện trong thời gian $O(n)$ bằng công thức:

$$b_1 = a_1 + a_2 + \cdots + a_k$$
$$b_{i+1} = b_i + a[i+k] - a[i], \forall i: 1 \le i < n-k+1$$

Tính mảng $f[1 \ldots n-k+1]$, ở đây $f[i]$ là giá trị lớn nhất $\in b[1 \ldots i]$. $O(n)$ bằng công thức:

$$f_1 = b_1$$
$$f_i = \max\{f_{i-1}, b_i\}, \forall i: 1 < i \le n-k+1$$

Tính mảng $g[1 \ldots n-k+1]$, ở đây $g[j]$ là giá trị lớn nhất $\in [j \ldots n-k+1]$. $O(n)$ bằng công thức:

$$g[n-k+1] = b[n-k+1]$$
$$g[j] = \max\{g[j+1], b[j]\}, \forall j: 1 \le j < n-k+1$$

Dễ thấy rằng khi Cám chọn k món quà liên tiếp tính từ vị trí i thì Tấm có hai lựa chọn:

Hoặc chọn k món quà liên tiếp đứng trước vị trí i, giá trị lớn nhất có thể thu được là $f[i-k]$

Hoặc chọn k món quà liên tiếp đứng sau vị trí $i+k-1$, giá trị lớn nhất có thể thu được là $g[i+k]$

Vậy Cám cần chọn vị trí i mà $\max\{f[i-k], g[i+k]\}$ bé nhất.

Đáp số có thể tính trong thời gian $O(n)$:

$$\min_{\forall i=1\ldots n}(\max\{f[i-k], g[i+k]\})$$

(Trong công thức này, các giá trị ứng với chỉ số nằm ngoài phạm vi mảng coi như bằng 0)

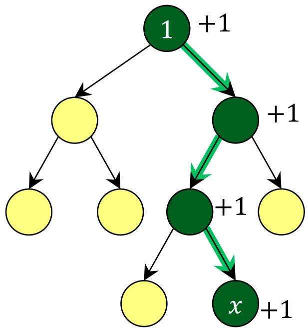
Bài 2. ĐIỀU CHỈNH CÂY (Thầy Nguyễn Thanh Hùng)

Gọi cây đã cho là A (ứng với dãy A), xét cây B (ứng với dãy B), gọi độ xấu của cây B là tổng chênh lệch giá trị giữa từng cặp nút tương ứng trên hai cây A và B ($\sum_{i=1}^{n}|a_i - b_i|$). Cây kết quả là cây có độ xấu thấp nhất và độ xấu đó chính là số đưa ra output.

Không khó để xây dựng thuật toán quy hoạch động tính hàm mục tiêu $f(i,k)$ là độ xấu tối thiểu của nhánh i nếu trong nhánh đó chọn đúng k nút lá mang số 1. Trong trường hợp là cây nhị phân, nhờ kỹ thuật đánh giá độ phức tạp chặt chẽ, có thể chỉ ra rằng thuật toán có độ phức tạp $O(n^2)$.

Ta sẽ trình bày một thuật toán tham lam trong trường hợp cây tổng quát, cài đặt dễ dàng với độ phức tạp $O(n^2)$ và có thể giảm cấp phức tạp xuống nữa bằng các cấu trúc dữ liệu đặc biệt.

Ý tưởng



Khởi tạo cây B mà tất cả các nút chứa số 0. Với x là một lá mang số 0, xét lệnh $Augment(x)$: Tăng các giá trị $b[.]$ dọc trên đường đi từ gốc đến lá x lên 1 đơn vị (bao gồm cả $b[1]$ và $b[x]$). Tại mỗi bước lặp, ta chọn lá x mang số 0 mà lệnh $Augment(x)$ cải thiện cây B được nhiều nhất (tức là làm giảm độ xấu của cây B đi nhiều nhất) và thực hiện luôn lệnh $Augment(x)$. Khi không có lệnh $Augment(x)$ nào có thể làm giảm độ xấu của cây B, thuật toán dừng và in kết quả.

Tính đúng đắn của giải thuật:

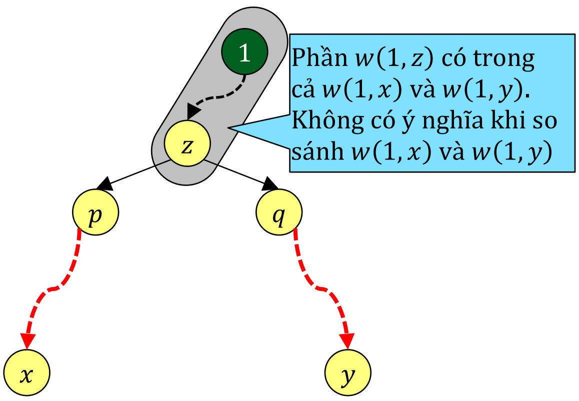
Vì lệnh $Augment(x)$ chỉ được thực hiện tại lá x mang số 0, một dãy lệnh $Augment(x)$ sẽ tạo ra cây B thỏa mãn điều kiện: Nút lá chứa số $\in \{0,1\}$ và giá trị tại mỗi nút nhánh bằng tổng giá trị các nút con. Ngoài ra có thể thấy rằng cây B cần tìm được tạo thành từ một dãy hữu hạn các lệnh $Augment(x)$. Việc của chúng ta chỉ là tìm ra dãy lệnh đó thôi.

Xét cây B tại một thời điểm nào đó, một nút i trên cây B được gán trọng số $+1$ nếu $b_i < a_i$ và trọng số -1 nếu $b_i \geq a_i$. Gọi $w(s,t)$ là trọng số đường đi từ nút s tới nút t (bằng tổng trọng số các nút trên đường đi). Giá trị $w(1,x)$ cho biết nếu thực hiện tiếp phép $Augment(x)$ thì độ xấu của cây B giảm đi bao nhiêu. Lý do đơn giản là nếu $b_i < a_i$ thì tăng b_i lên 1 sẽ làm cho độ chênh lệch $|a_i - b_i|$ giảm 1, còn nếu $b_i \geq a_i$ thì tăng b_i lên 1 sẽ làm độ chênh lệch $|a_i - b_i|$ tăng 1. Mức giảm độ chênh lệch được phản ánh qua trọng số trên các nút.

Nhận xét: Tại mỗi bước lặp, xét lá x có $b[x] = 0$ và $w(1,x)$ dương và lớn nhất, khi đó $Augment(x)$ chắc chắn được chọn vào một phương án tối ưu.

Chứng minh

Với một tình trạng hiện tại của cây B, gọi x là lá mang số $b[x] = 0$ và có $w(1,x) > 0$ lớn nhất. Trước hết thấy rằng nếu được thực hiện luôn, lệnh $Augment(x)$ chắc chắn làm giảm độ xấu cây B. Xét một dãy $Augment(.)$ không chứa phép $Augment(x)$, trong dãy này chọn y là lá thỏa mãn $z = LCA(x,y)$ ở sâu nhất, ta sẽ chỉ ra rằng nếu thay lệnh $Augment(y)$ bởi $Augment(x)$ ta sẽ được dãy lệnh không tệ hơn dãy lệnh đang có.



Vì kết quả dãy lệnh $Augment(.)$ không phụ thuộc thứ tự, ta có thể coi lệnh $Augment(y)$ nằm cuối dãy và tạm thời chưa thực hiện lệnh đó, để chứng minh lựa chọn $Augment(x)$ tốt hơn $Augment(y)$ ở bước cuối, ta sẽ chỉ ra $w(1,x) \geq w(1,y)$. Điều này tương đương với $w(p,x) \geq w(q,y)$ với p và q là con của z và lần lượt là tiền bối của x và y (xem hình bên)

Khi chưa thực hiện lệnh $Augment(.)$ nào, $w(p,x) \geq w(q,y)$ (do cách chọn lá x có $w(1,x) \geq w(1,y)$). Dãy các lệnh $Augment(.)$ không làm thay đổi trọng số các đỉnh trên đường đi từ p tới x (do cách chọn có nút $z = LCA(x,y)$ sâu nhất). Dãy các lệnh $Augment(.)$ có thể làm thay đổi trọng số các đỉnh trên đường đi từ q tới y nhưng đều là phép đổi trọng số +1 thành -1 (giảm trọng số), tức là $w(p,x) \geq w(q,y)$ sau dãy lệnh $Augment(.)$. Điều này cho thấy ở bước cuối cùng, lệnh $Augment(x)$ cải thiện cây B không tệ hơn lệnh $Augment(y)$, Khi đã biết chắc chắn $Augment(x)$ được chọn vào phương án tối ưu ta có thể thực hiện nó luôn. Lập luận tương tự, với tình trạng mới của cây B, ta lại chọn lá x mang số $b[x] = 0$, có trọng số $w(1,x)$ dương và lớn nhất...

Thiết kế

Vì lệnh $Augment(x)$ tăng các giá trị $b[.]$ dọc đường đi từ 1 tới x lên 1, ta có thể thực hiện thao tác tương đương: Giảm các giá trị $a[.]$ dọc đường đi từ 1 tới x đi 1, còn các giá trị $b[.]$ vẫn giữ bằng 0.

Khởi tạo: S là tập các lá.

Lặp:

- Tính các trọng số $w(1,.)$ trên các lá $\in S$. Điều này có thể thực hiện bằng BFS hoặc DFS. $O(n)$.
- Xác định $x \in S$ có $w(1,x)$ lớn nhất. $O(|S|)$
  - Nếu $w(1,x) \le 0$, dừng vòng lặp
  - Nếu $w(1,x) > 0$, giảm các $a[.]$ trên đường đi từ $1 \rightsquigarrow x$ đi 1, loại bỏ lá x khỏi S. $O(n)$

Output: $\sum_{i=1}^{n}|a_i|$

Với cách cài đặt này nhãn $b[.]$ coi như luôn = 0, không cần khai báo. Nút có $a[i] > 0$ mang trọng số $+1$, nút có $a[i] \le 0$ mang trọng số -1.

Độ phức tạp $O(|S|.n)$ tức là $O(n^2)$.

Thuật toán có thể cải tiến xuống ĐPT $O(n\log^2 n)$ khi dùng các CTDL cho phép thực hiện nhanh các lệnh $Augment(.)$ và tìm nhanh được lá k có $w(1,k)$ lớn nhất.

Bài 3. CHUYỂN SỎI (Thầy Lê Minh Hoàng)

Đặt q và r lần lượt là thương và số dư của phép chia $\sum_{i=1}^{n} a_i$ cho n. Có thể thấy rằng ở trạng thái cuối cùng, mỗi đống sỏi có thể có q hoặc $q+1$ viên ($n-r$ đống có q viên và r đống có $q+1$ viên). Mặt khác có thể trừ tất cả các a_i đi cùng một hằng số mà không ảnh hưởng tới kết quả, ta thực hiện $a_i -= q, \forall i$. Bây giờ các đống sỏi có thể có số sỏi âm, luật chuyển không thay đổi ($a_i \pm= 1; a_{i+1} \mp= 1$), trạng thái cuối cùng mỗi đống sỏi sẽ có 0 hoặc 1 viên (có đúng r đống 1 viên, còn lại là các đống 0 viên).

Đặt $b_i = a_1 + a_2 + \cdots + a_i$, khi đó:

- Việc chuyển 1 viên sỏi từ $i \to i+1$ tức là $a_i -= 1; a_{i+1} += 1$, điều này tương đương với $b_i -= 1$.
- Việc chuyển 1 viên sỏi từ $i+1 \to i$ tức là $a_i += 1; a_{i+1} -= 1$, điều này tương đương với $b_i += 1$.
- Ở trạng thái cuối cùng $b_i - b_{i-1} = a_i \in \{0,1\}$, tức là dãy B tăng dần và phần tử sau hơn phần tử trước không quá 1 đơn vị. Ngoài ra $b_n = \sum_{i=1}^{n} a_i = r$

Vậy thì bài toán có thể phát biểu theo cách khác:

Cho dãy $B = (0 = b_0, b_1, b_2, \dots, b_n = r)$, mỗi phép biến đổi tăng/giảm một phần tử đúng 1 đơn vị, cần dùng ít phép biến đổi nhất để biến dãy B thành dãy C thỏa mãn:

- $c_0 = 0$;
- $c_n = r$;
- $0 \le c_i - c_{i-1} \le 1, \forall i = 1,2,\dots,n$;

Có thể dùng thuật toán quy hoạch động với độ phức tạp $O(n.r)$: Tính hàm mục tiêu $f(n,r)$ bằng số phép biến đổi ít nhất để biến dãy $b_0 \dots b_n$ thành dãy $c_0 \dots c_n$ thỏa mãn $c_0 = 0, c_n = r$ và $0 \le c_i - c_{i-1} \le 1$ (Trường hợp $r = 0$, subtask 1, có thể biến đổi về giải thuật ad-hoc). Tuy nhiên giải pháp này rất khó cải tiến về độ phức tạp tính toán. Ta sẽ trình bày một giải pháp tham lam khá dễ cài đặt, và có thể cải tiến giảm độ phức tạp bằng các cấu trúc dữ liệu phù hợp.

Ý tưởng

Gọi độ xấu của dãy C là số phép biến đổi ít nhất để biến dãy B thành dãy C ($\sum_{i=0}^{n}|b_i - c_i|$). Xét lệnh $Add(i)$: Tăng $c[i \dots n]$ lên 1 đơn vị ($1 \le i \le n$). Chú ý rằng tham số của lệnh $Add(.)$ là số dương, lệnh $Add(0)$ là không hợp lệ.

Khởi tạo dãy C toàn số 0 và lặp r bước. Tại mỗi bước, chọn một vị trí i mà lệnh $Add(i)$ cải thiện dãy C được nhiều nhất (tức là làm giảm độ xấu của dãy C đi nhiều nhất) và thực hiện luôn lệnh $Add(i)$, đồng thời đánh dấu vị trí i không cho chọn nữa. Sau r bước lặp, đáp số là độ xấu của cây C.

Tính đúng đắn của giải thuật

Dễ thấy rằng một bộ r lệnh $Add(i)$ với các tham số i dương và hoàn toàn phân biệt sẽ tạo ra dãy C thỏa mãn: $c_0 = 0$, $c_n = r$ và $0 \le c_i - c_{i-1} \le 1$. Dãy C mà chúng ta muốn biến đổi thành cũng có thể tạo thành từ bộ r lệnh $Add(.)$ với các tham số hoàn toàn phân biệt. Việc của chúng ta chỉ là tìm bộ r lệnh để tạo ra dãy C có độ xấu thấp nhất, gọi là bộ r lệnh tối ưu.

Vì giải thuật luôn giữ $c_0 = b_0 = 0$, ta bỏ qua chỉ số 0 trong hai dãy B, C. Với một tình trạng của dãy C, đặt trọng số tại i bằng $+1$ nếu $c_i < b_i$ và đặt trọng số tại i bằng -1 nếu $c_i \ge b_i$. Ký hiệu $w(i,j)$ là tổng trọng số các điểm từ i tới j. Nếu thực hiện $Add(k)$, độ xấu của dãy C sẽ giảm đi $w(k,n)$, điều này dễ hiểu vì khi $c_i < b_i$, việc tăng c_i lên 1 sẽ làm độ chênh lệch $|b_i - c_i|$ giảm đi 1, còn khi $c_i \ge b_i$, việc tăng c_i lên 1 sẽ làm độ chênh lệch $|b_i - c_i|$ tăng lên 1. Mức giảm độ chênh lệch được phản ánh qua trọng số.

Nhận xét: Trước mỗi bước lặp, xét các vị trí mà tại đó chưa thực hiện lệnh $Add(.)$ nào, chọn vị trí k có $w(k,n)$ lớn nhất, khi đó $Add(k)$ chắc chắn được chọn vào một phương án tối ưu.

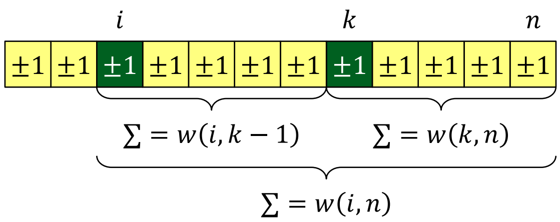
Chứng minh

Tại thời điểm khởi đầu với dãy $C = 0$. Gọi k là điểm có $w(k,n)$ lớn nhất, xét một bộ gồm r lệnh $Add(.)$ tối ưu mà không chứa lệnh $Add(k)$. Chọn lệnh $Add(i)$ trong bộ lệnh có i đứng trước k và gần k nhất. Nếu $k < \forall i$ trong bộ lệnh, chọn i đứng sau k và gần k nhất.

Vì thứ tự thực hiện các lệnh $Add(.)$ không quan trọng, ta thực hiện trước tất cả các lệnh $Add(.)$ trong bộ lệnh trừ lệnh $Add(i)$, đến đây ta sẽ chỉ ra rằng thực hiện nốt lệnh cuối cùng là $Add(k)$ sẽ thu được kết quả ít ra là không tệ hơn so với việc chọn $Add(i)$.

Như đã phân tích, nếu ta thực hiện $Add(k)$, độ xấu của dãy C giảm đi $w(k,n)$ còn nếu thực hiện $Add(i)$, độ xấu của dãy C giảm đi $w(i,n)$.

Nếu i đứng trước k:



Tại thời điểm khởi đầu với dãy $C = 0$, do $w(k,n)$ lớn nhất nên

$$\underbrace{w(i,n) - w(k,n)}_{w(i,k-1)} \le 0$$

Các lệnh $Add(.)$ chỉ làm giảm trọng số tại các vị trí (Biến $+1$ thành -1) nên ta vẫn có $w(i,k-1) \le 0$. Vậy trước khi làm lệnh $Add(.)$ cuối cùng:

$$w(i,n) \le w(k,n)$$

Nếu i đứng sau k:

$$\pm1\ \pm1\ \underbrace{\underbrace{\overset{k}{\pm1}\ \pm1\ \pm1\ \pm1\ \pm1}_{\sum = w(k,i-1)}\ \underbrace{\overset{i}{\pm1}\ \pm1\ \pm1\ \pm1\ \overset{n}{\pm1}}_{\sum = w(i,n)}}_{\sum = w(k,n)}$$

Tại thời điểm trước bước lặp đang xét, do $w(k,n)$ lớn nhất nên

$$\underbrace{w(k,n) - w(i,n)}_{w(k,i-1)} \geq 0$$

Như quy ước về cách chọn i, trường hợp này chỉ xảy ra khi không có lệnh $Add(.)$ nào trong bộ lệnh thực hiện tại vị trí đứng trước k, trọng số các điểm từ k tới $i-1$ không đổi qua các lệnh $Add(.)$ nên $w(k,i-1)$ không đổi ≥ 0. Suy ra trước lệnh $Add(.)$ cuối cùng:

$$w(k,n) \geq w(i,n)$$

Nếu $w(k,n) > w(i,n)$, việc thay $Add(i)$ bởi $Add(k)$ sẽ cho dãy kết quả C tốt hơn, mâu thuẫn với giả thiết bộ lệnh đang chọn là tối ưu. Vậy thì $w(k,n) = w(i,n)$, việc thay $Add(i)$ bởi $Add(k)$ vẫn cho một phương án tối ưu khác.

Khi đã biết chắc lệnh $Add(k)$ được chọn ta có thể thực hiện luôn nó để tăng các phần tử từ c_k tới c_n lên 1. Lặp lại chứng minh trên với tình trạng hiện tại của dãy C và vị trí k được đánh dấu để không chọn lại nữa, chọn phần tử k' có $w(k',n)$ lớn nhất và dùng lệnh $Add(k')$ thay thế cho một lệnh $Add(.)$ trong số $r-1$ lệnh $Add(.)$ còn lại...

Thiết kế

Lệnh $Add(k)$ tăng $c[k \dots n]$ lên 1 đơn vị, ta có thể làm một cách khác tương đương: Giảm $b[k \dots n]$ đi 1 đơn vị, còn dãy C vẫn luôn giữ $= 0$.

Giải thuật: lặp r lần:

- Tính các trọng số $w(i,n)$, tìm k có $w(k,n)$ lớn nhất. $O(n)$
- Giảm $b[k \dots n]$ đi 1, đánh dấu vị trí k không cho chọn nữa. $O(n)$

Với cách cài đặt này dãy C luôn $= 0$, không cần khai báo. Điểm i có trọng số $+1$ nếu $b_i > 0$ và trọng số -1 nếu $b_i \leq 0$.

Giảm độ phức tạp

Để giảm độ phức tạp, cần phải thực hiện tốt hai việc:

- Khi giảm các $b[k \dots n]$ đi 1, cần lọc nhanh những $b[.]$ nào chuyển từ 1 thành 0 để đổi trọng số vị trí đó từ $+1$ thành -1. (Dùng Segment Trees để giảm $b[k \dots n]$ và lấy min trong các $b[.] \geq 0$). $O(\log n)$
- Xác định điểm k có trọng số $w(k,n)$ lớn nhất. (Dùng Segment Trees). $O(\log n)$

ĐPT $O(n \log n)$

Có một điều thú vị (Lúc làm đề vội quá không nghĩ ra) là nếu yêu cầu giải bài 2 với độ phức tạp $O(n \log^2 n)$ thì bài 3 chỉ là một subtask của bài 2 😊.