

Chia để trị (Divide and Conquer)

Tác giả:

- ▶ Nguyễn Đức Kiên, Trường Đại học Công nghệ, ĐHQGHN

Reviewer:

- ▶ Nguyễn Hoàng Vũ - Trường Đại học Công nghệ, ĐHQGHN
- ▶ Nguyễn Minh Nhật - THPT chuyên Khoa học Tự nhiên, ĐHQGHN

Mở đầu

Chia để trị (Divide and Conquer, DnC) chỉ việc chia nhỏ bài toán thành các phần nhỏ có cùng dạng với nó, tới khi có thể giải được một cách dễ dàng (thường là có ngay kết quả), sau đó dùng các kết quả này kết hợp lại để giải được bài toán lớn. Các bước để giải một bài toán chia để trị bao gồm:


- ▶ Chia bài toán thành các bài toán con (thường là chia đôi hoặc xấp xỉ như vậy).
- ▶ Giải các bài toán con. Nếu vẫn gặp khó khăn với các bài toán con, ta lại tiếp tục chia nó thành các bài toán nhỏ hơn cho tới khi dễ dàng tìm được kết quả.
- ▶ Tổng hợp kết quả các bài toán con lại để có kết quả của bài toán lớn nếu cần.

Bạn đọc có thể thấy những bước trên có phần tương đồng với quá trình làm các bài toán đệ quy cơ bản. Các bài toán đó cũng có thể coi là chia để trị, và đệ quy cũng là một cách để cài đặt các thuật chia để trị.

Tư tưởng chia để trị cũng xuất hiện rất đa dạng và phổ biến trong các bài toán, thuật toán và cấu trúc dữ liệu: tìm kiếm nhị phân, cấp trúc cây tìm kiếm nhị phân (BST), heap, cây phân đoạn (segment tree), cây BIT/Fenwick, mảng thưa, các thuật toán sắp xếp, lũy thừa nhanh, ...

Xác định độ phức tạp

Ta nhắc lại một vài ký hiệu trước khi vào phần này:

- ▶ $\log_a b$: Logarit cơ số a của b , là số thực k thỏa mãn $a^k = b$. Bài viết này sẽ sử dụng ký hiệu $\log b$ cho trường hợp $a = 2$.
- ▶ $T(n)$: Thời gian chạy thuật toán với kích thước dữ liệu đầu vào là n , tính bằng số phép tính.
- ▶ $O()$ (big-O): Các hàm độ phức tạp thuật toán. Chi tiết bạn đọc tham khảo [Độ phức tạp thời gian](#) 

Chi tiết về các ký hiệu độ phức tạp và thời gian bạn đọc có thể tham khảo bài .

Định lý Thợ (Master theorem)

Các công thức dưới đây được nhắc đến trong sách *Introduction to Algorithms* (xem phần Tài liệu tham khảo). Chưa rõ tên tiếng Việt của thuật toán có nguồn gốc từ đâu.

Độ phức tạp của các thuật toán chia để trị cài đặt dưới dạng đệ quy được xác định bằng **Định lý Thợ**.

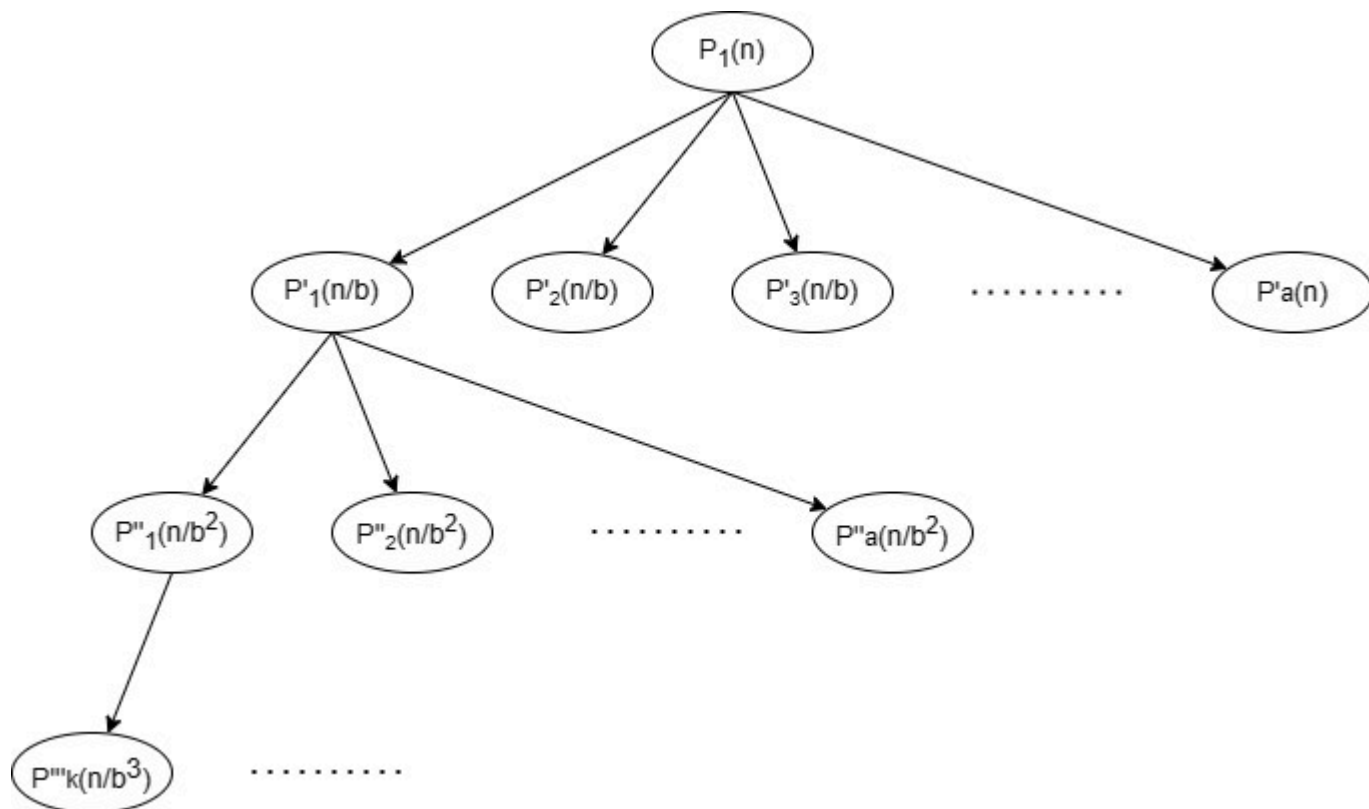
Giả sử có một bài toán P với dữ liệu đầu vào có độ lớn là n (mảng có độ dài n chẳng hạn) được cài đặt bằng đệ quy, tại mỗi vòng đệ quy bài toán được chia thành a bài toán con như sau:

```

1  void P(<input kích thước n>)
2  {
3      if (n < k)  //k là một hằng số
4      {
5          <giải bài toán trực tiếp (base case)>
6      }
7      else
8      {
9          for (int i = 1; i <= a; i++)
10         {
11             P(<input kích thước n/b>);
12         }
13         <kết hợp kết quả các bước trên>
14     }
15 }

```

Nếu ta coi mỗi bài toán con là một nút trên một cây và mỗi lần gọi đệ quy từ một bài toán con ta nối thêm một nút con vào nút biểu diễn bài toán nói trên thì bài toán lớn của chúng ta sẽ có dạng như sau:



Tại mỗi nút của cây trên, nếu việc kết hợp kết quả các bài toán con mất $f(n)$ thời gian, thì thời gian chạy tại một nút với kích thước dữ liệu là n có thể được tính theo công thức truy hồi:

$$T(n) = \begin{cases} a \times T(\frac{n}{b}) + f(n) & \text{khi } n \geq k \\ O(1) & \text{khi } n < k \end{cases}$$

với k là một hằng số nào đó, tùy thuộc vào thuật toán.

Ví dụ, với thuật toán MergeSort, tại mỗi bước ta chia một đoạn có độ dài n thành hai đoạn con có độ dài $n/2$ hoặc xấp xỉ số đó. Thuật toán sẽ có thời gian chạy là $T(n) = 2T(\frac{n}{2}) + O(n)$ khi $n > 1$ và $O(1)$ khi $n = 1$.

Bây giờ, ta lại xét giá trị $f(n)$. Giả sử $f(n)$ viết được dưới dạng $O(n^p \log^q n)$ (Định lý Thợ chỉ áp dụng khi $f(n)$ có độ phức tạp đa thức). Chúng ta có thể tiếp tục thu gọn biểu thức như sau:

Quan hệ a và b^p	Biểu thức $T(n)$
$a > b^p$	$T(n) = O(n^{\log_b a})$
$a = b^p$	$T(n) = O(n^{\log_b a} \log^{q+1} n)$ khi $q > -1$ $T(n) = O(n^{\log_b a} \log \log n)$ khi $q = -1$ $T(n) = O(n^{\log_b a})$ khi $q < -1$
$a < b^p$	$T(n) = O(n^p \log_q n)$ khi $q \geq 0$ $T(n) = O(n^p)$ khi $q < 0$

Một số ví dụ:

- Thuật toán Tìm kiếm nhị phân (Binary search) mỗi lần chia bài toán thành 2 phần bằng nhau nhưng chỉ xét 1, không cần kết hợp kết quả sẽ có $T(n) = T(n/2) + O(1)$. Thời gian chạy trung bình của thuật toán là $T(n) = O(\log n)$, ứng với $a = 1, b = 2, p = 0, q = 0$.
- Thuật toán MergeSort chia đôi dãy hiện tại thành 2 phần bằng nhau, lấy cả 2 và phải xét lại cả 2 phần để lấy kết quả sẽ có $T(n) = 2T(n/2) + O(n)$. Thời gian chạy trung bình của thuật toán là $T(n) = O(n \log n)$, ứng với $a = 2, b = 2, p = 1, q = 0$ (Chi tiết về thuật MergeSort bạn đọc có thể đọc phần dưới).
- Một thuật toán chia để trị có $T(n) = 3T(n/2) + \log^2 n$. Thời gian chạy trung bình của thuật toán là $T(n) = O(n^{\log_2 3})$, tương ứng với $a = 3, b = 2, p = 0, q = 2$.

Định lý Thợ là một công cụ hữu hiệu để xác định độ phức tạp của một thuật toán chia để trị. Chỉ cần xác định được số bài toán con, kích thước dữ liệu các bài toán con và độ phức tạp của việc kết hợp dữ liệu, ta dễ dàng tìm ra độ phức tạp chung của bài toán.


Một số ví dụ áp dụng Chia để trị

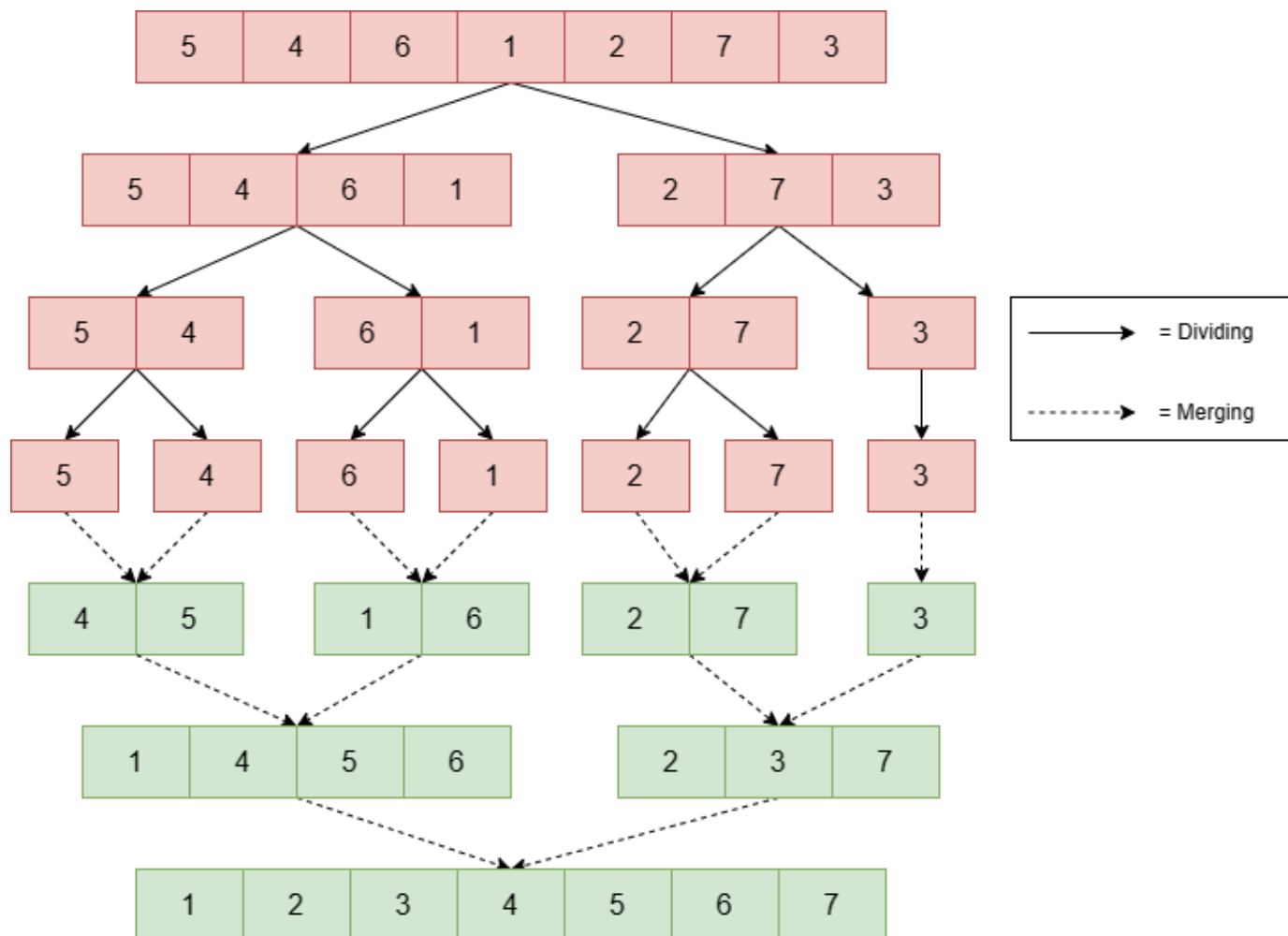
Các bài toán áp dụng Chia để trị chỉ có chung một phương pháp như đã nói ở trên. Khi sử dụng, thứ mà ta quan tâm chủ yếu là độ phức tạp; thông thường, chia để trị là công cụ tối ưu độ phức tạp khá hiệu quả.

MergeSort

Đề bài: Sắp xếp một dãy gồm n số nguyên ($n \leq 10^6$).

Phân tích thuật toán

Đây là một thuật toán sắp xếp nổi tiếng và cũng hay được áp dụng (nếu không được phép sử dụng các thư viện có sẵn). Thuật toán này sử dụng phương pháp đệ quy. Tại mỗi vòng đệ quy, giả sử đang cần sắp xếp một đoạn $[l, r]$ ta chia dãy làm hai phần bằng nhau, $[l, mid]$ và $[mid + 1, r]$ với $mid = \left\lfloor \frac{l + r}{2} \right\rfloor$. Sau khi đã gọi đệ quy các đoạn con, ta tiến hành hợp nhất hai đoạn này. Việc hợp nhất hai đoạn đã sắp xếp được tiến hành bằng phương pháp [hai con trỏ](#) , có độ phức tạp là $O(n)$. Trong trường hợp một đoạn chỉ có một phần tử duy nhất, ta coi như nó đã được sắp xếp.



Cài đặt

► [Nhấn để hiện code](#)

Đánh giá

Theo như phân tích độ phức tạp đã đề cập ở trên, độ phức tạp trung bình của MergeSort là $O(n \log n)$. Thực tế thì trong mọi trường hợp, độ phức tạp của MergeSort luôn là $O(n \log n)$.

Cập điểm gần nhất

Đề bài: Cho n điểm trên mặt phẳng ($2 \leq n \leq 10^5$). Hãy tìm khoảng cách nhỏ nhất giữa hai điểm bất kỳ trong đó.

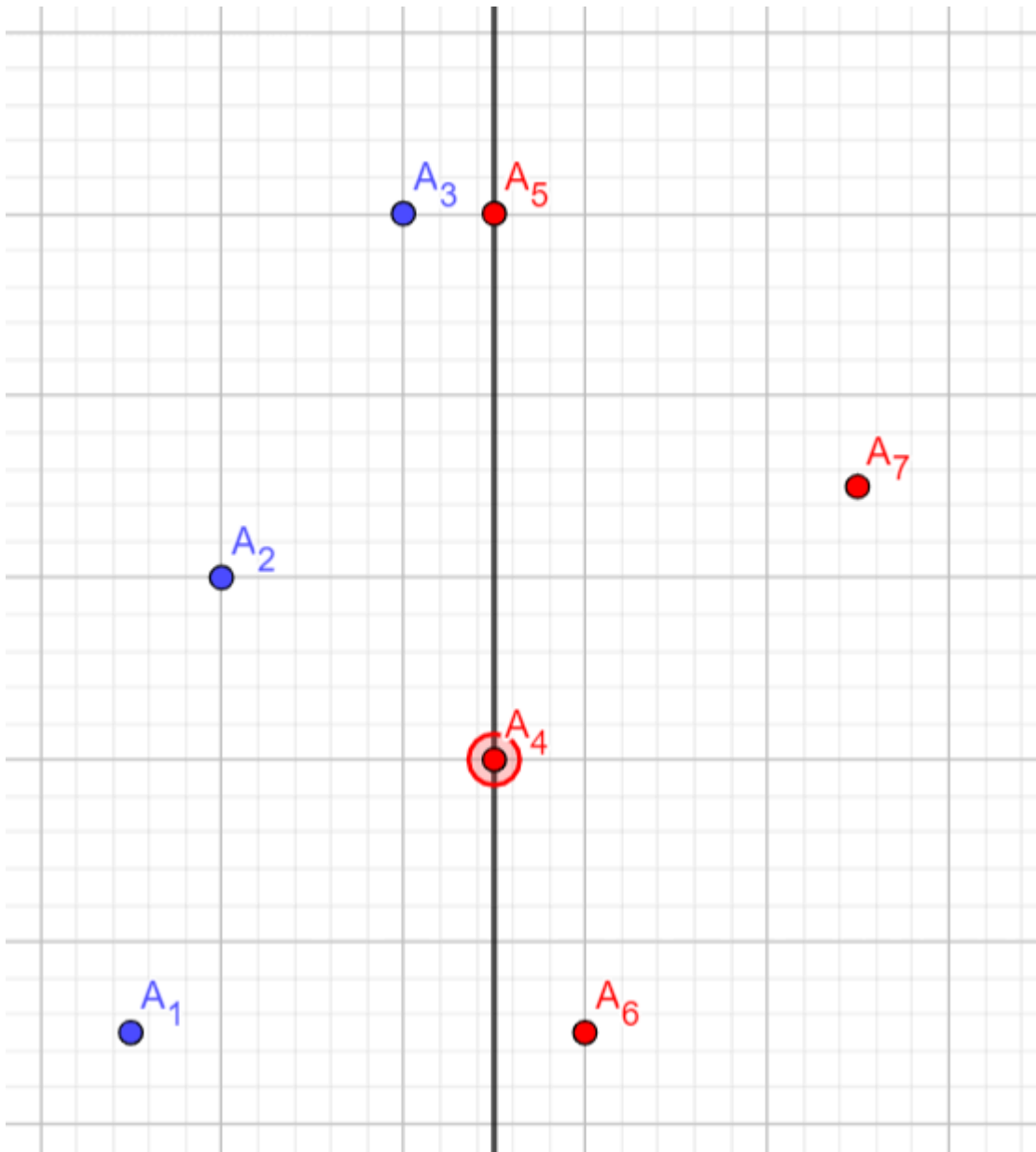
Đề bài VNOI: [NEAREST](#) 

Phân tích

Giả sử có n điểm A_1, A_2, \dots, A_n .

Ta có thể sử dụng một thuật toán "ngây thơ" cho bài này: xét tất cả mọi cặp điểm, kiểm tra xem khoảng cách giữa hai cặp điểm nào là gần nhau nhất. Độ phức tạp khi đó sẽ là $O(n^2)$ trong mọi trường hợp, chưa đủ để vượt qua giới hạn của bài toán này.

Ta nghĩ đến việc sử dụng chia để trị. Trước hết, ta sắp xếp các điểm trong tập hợp theo hoành độ x . Tại mỗi vòng đệ quy, ta chia tập điểm hiện tại thành hai phần bên trái và bên phải vị trí mid ta chọn. Base case (trường hợp cơ bản) lúc này thay vì là $l = r$ thì sẽ là $r - l \leq 2$, do ta không thể xác định khoảng cách với 1 điểm, và cũng cần đảm bảo rằng khi chạy đệ quy không tồn tại tập nào có độ lớn như vậy. Ngoài trường hợp đó, ta thu được kết quả của 2 tập trái và phải. Tuy nhiên, việc kết hợp kết quả không đơn giản, vì một điểm ở bên trái A_{mid} vẫn có thể tạo ra khoảng cách ngắn nhất với một điểm bên phải. Ta cũng không thể chạy hết từng cặp điểm một trong hai tập này, vì khi đó theo Định lý Thợ độ phức tạp trung bình sẽ lên đến $O(n^2)$.



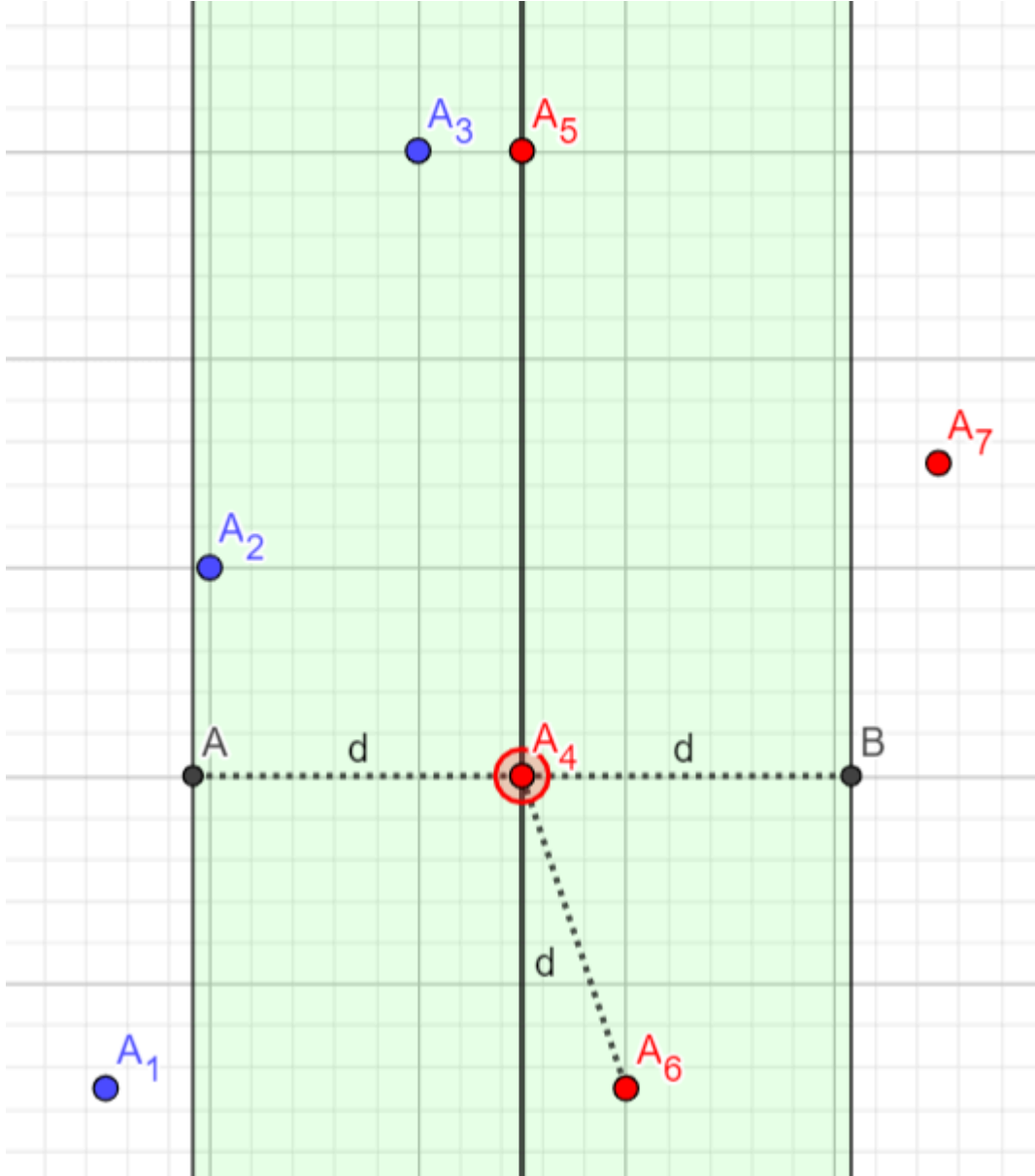
Ở hình vẽ trên, hai màu xanh và đỏ tượng trưng cho hai nửa phải và trái. Điểm A_4 đóng vai trò là A_{mid} , thuộc tập bên phải.

Gọi d là giá trị nhỏ hơn giữa khoảng cách ngắn nhất giữa hai điểm ta vừa thu được ở tập bên phải và tập bên trái. Cụ thể:

$$d = \min\{nearest(l, mid), nearest(mid + 1, r)\}$$

Khi đó, trong cùng một tập hợp, không tồn tại một cặp điểm nào có khoảng cách ngắn hơn d . Giữa hai tập hợp lúc này ta sẽ chỉ quan tâm đến các cặp điểm có khoảng cách nhỏ hơn d .

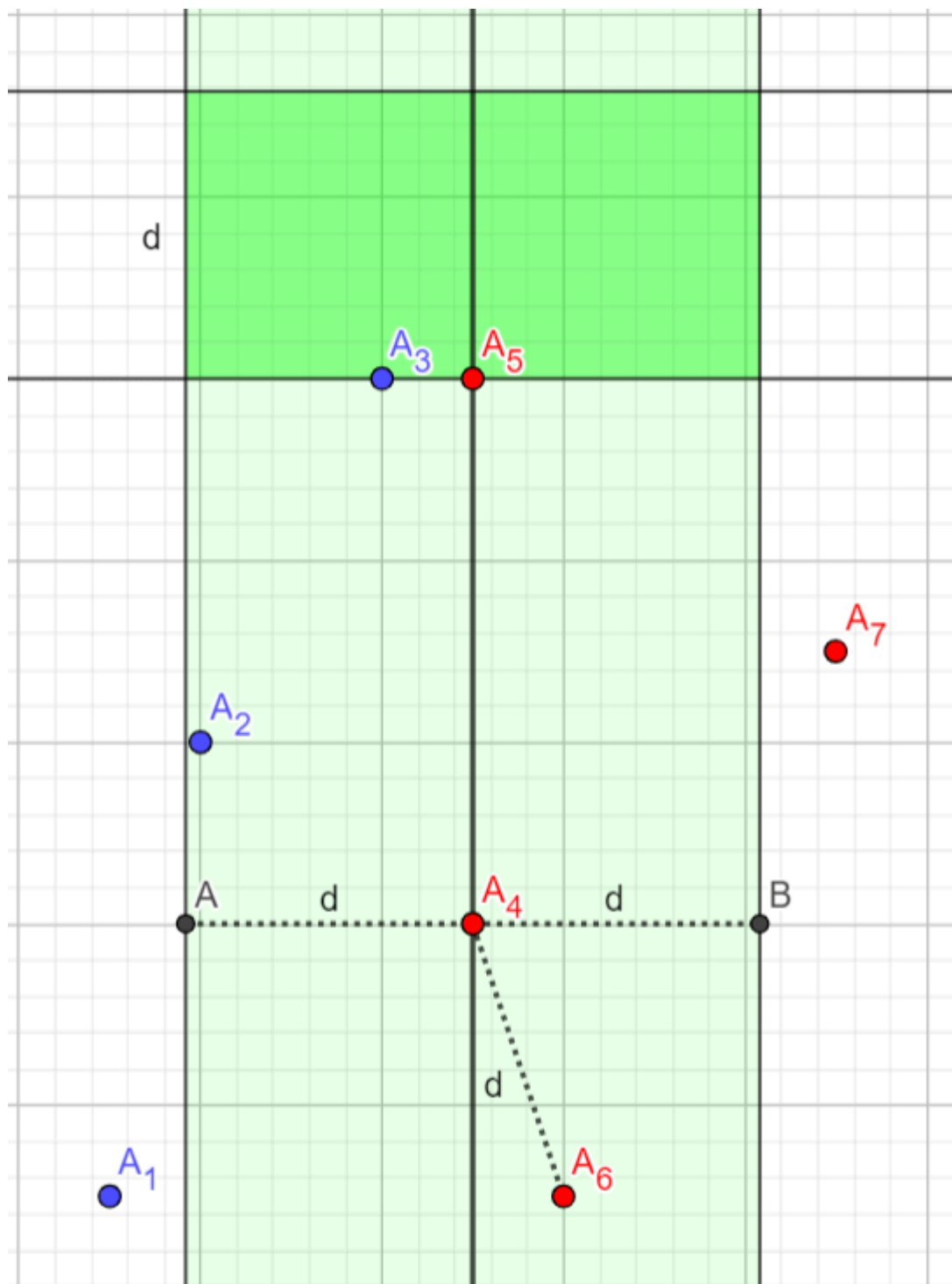
Xét các điểm có hoành độ cách A_{mid} một khoảng không vượt quá d . Các điểm này nằm giữa các đường thẳng $x = x_{mid} - d$ và $x = x_{mid} + d$:



Đến đây, ta có một nhận xét quan trọng: Với mỗi điểm A_m nằm trong miền nằm giữa hai đường thẳng nói trên (vùng được tô màu), tồn tại không quá 7 điểm khác A_m có tung độ y lớn hơn không quá d so với y_m .

► Chứng minh (nhấn để hiện)

Nếu ta sắp xếp các điểm trong miền này theo thứ tự y tăng dần, với một điểm bất kỳ ta chỉ cần xét một số điểm lân cận thỏa mãn chênh lệch tung độ không vượt quá d , rồi tính khoảng cách giữa chúng.



Khi cài đặt, sau khi tiến hành tìm khoảng cách ngắn nhất giữa hai điểm ta có thể giữ nguyên trạng thái sau khi sắp xếp theo y của đoạn đó, rồi dùng phép `merge()` như bài MergeSort ở trên để sắp xếp nhanh đoạn lớn trong $O(n)$.

Cài đặt


► [Nhấn để hiện code](#)

Đánh giá

Mỗi tập $[l, r]$ được chia thành hai tập con, mỗi tập con có bộ dữ liệu bằng đúng một nửa tập lớn. Việc tìm kết quả của đoạn lớn bao gồm việc ghép đoạn để sắp xếp lại mất $O(n)$ và tính khoảng cách nhỏ nhất giữa các điểm ở giữa hết $O(7n) = O(n)$. Do vậy thuật này có $T(n) = 2T(\frac{n}{2}) + O(n)$ và có độ phức tạp trung bình

là $O(n \log n)$ theo Định lý Thợ. Trong mọi trường hợp, thuật toán đều thực hiện những bước tương tự và độ phức tạp là $O(n \log n)$.

Bonus

Bài toán này còn một lời giải khác bằng cách sử dụng kỹ thuật đường quét (sweep-line) kết hợp với cấu trúc [set](#). Lời giải này có cùng độ phức tạp nhưng ngắn gọn và dễ cài đặt hơn lời giải bằng chia để trị. Bạn đọc tự cài đặt và tìm hiểu thêm, tham khảo [code của skyvn97](#) .

Bài toán Truy vấn trên mảng cố định

Bài toán

Bài toán **Truy vấn trên mảng cố định (SRQ - Static Array Queries)** được mô tả như sau:

Xét phép toán bất kỳ \star và mảng a gồm các số a_1, a_2, \dots, a_n . Ta phải trả lời q truy vấn, mỗi truy vấn yêu cầu ta tính $a_l \star a_{l+1} \star \dots \star a_r$, với l, r là các giá trị cho trước, $l, r \in [1, n]$.

Trong trường hợp \star là phép toán có tính chất kết hợp, cụ thể hơn, phép toán này áp dụng được trên các giá trị (không nhất thiết là số) a, b, c sao cho $(a \star b) \star c = a \star (b \star c)$, ta có thể sử dụng chia để trị để giải. Một số ví dụ cho phép toán này là phép cộng, phép nhân, phép lấy min.

Thuật toán giải

Có rất nhiều cách giải bài toán này với độ phức tạp không gian và thời gian logarit tuyến tính ($O(n \log n)$), như cây phân đoạn, BIT, mảng thưa, ... Chia để trị cũng là một cách hiệu quả để giải quyết bài toán, đặc biệt trong trường hợp độ phức tạp cho mỗi truy vấn cần rất thấp.

Giả sử tất cả các truy vấn (lq, rq) đều thỏa mãn điều kiện $l \leq lq \leq rq \leq r$. Ban đầu, ta có $l = 1, r = n$. Đặt $mid = \lfloor \frac{l+r}{2} \rfloor$ (tùy trường hợp, để tính toán thuận lợi, mid có thể nhận các giá trị khác nhau, nhưng thường ta lấy vị trí chính giữa). Gọi $leftAcc[i]$ là "tổng" hậu tố tính từ mid tới i hay:

$$leftAcc[i] = a[i] \star a[i+1] \star \dots \star a[mid]$$

Tương tự, ta gọi $rightAcc[i]$ là "tổng" tiền tố tính từ $mid + 1$ tới i :

$$rightAcc[i] = a[mid+1] \star a[mid+2] \star \dots \star a[i]$$

Với các truy vấn thỏa mãn $lq \leq mid \leq rq$, kết quả của truy vấn đó sẽ là $leftAcc[i] \star rightAcc[i]$. Điều này là hiển nhiên theo tính chất kết hợp. Với các truy vấn còn lại, hiển nhiên chúng chỉ nằm hoàn toàn ở một trong hai bên của mid . Ta thay đổi các giá trị l, r và tính các giá trị $leftAcc, rightAcc$ theo l, r mới. Cứ như vậy tới khi $l = r$.

Về độ phức tạp, ở mỗi bước ta chia mảng độ dài n thành 2 phần đều nhau có kích thước dữ liệu là $\frac{n}{2}$. Sau khi có hai đoạn này, ta mất $O(n)$ để tính $leftAcc, rightAcc$, và $O(1)$ cho mỗi truy vấn, tổng cộng là $O(n + q)$. Độ phức tạp thời gian của thuật toán là $O((n + q) \log n)$, chạy ổn định, còn về không gian chỉ cần $O(n + q)$.

Hai mảng $leftAcc$ và $rightAcc$ không có chung nhau một vị trí nào, vậy nên ta có thể kết hợp lại thành một mảng acc . Khi đó với một đoạn $[l, r]$ có phần tử ở giữa là mid , các giá trị từ $acc[l]$ đến $acc[mid]$ tương đương với $leftAcc$, và các giá trị $acc[mid+1]$ đến $acc[r]$ tương đương với $rightAcc$.

Nếu phải sử dụng các truy vấn online, ta lưu lại acc của mỗi lần đệ quy dưới dạng một mảng hai chiều giống như mảng thưa, ví dụ $acc[j][i]$ là kết quả $acc[i]$ ở vòng đệ quy với độ sâu j (xem ví dụ để hiểu rõ hơn). Bằng

cách này, ta vẫn có thể trả lời mọi truy vấn trong $O(1)$, và độ phức tạp trở thành $O(n \log n + q)$ cho thời gian và $O(n \log n + q)$ cho không gian.

Ví dụ

Đề bài: [SEGPORD](#) 

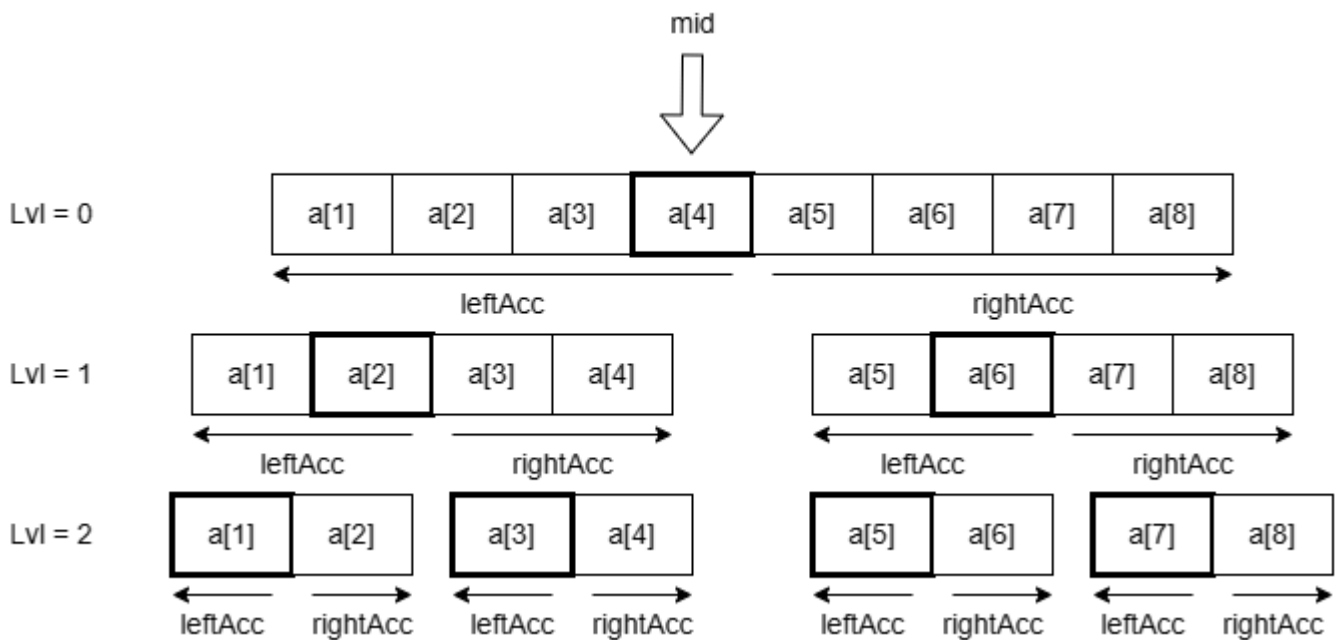
Tóm tắt: Cho dãy A gồm N số nguyên dương và một số nguyên dương P . Có $Q \leq 2 \times 10^7$ truy vấn, truy vấn thứ i yêu cầu tìm tích của các số A_j với $L_i \leq j \leq R_i$, lấy số dư khi chia cho P . Chú ý rằng các truy vấn phải được xử lý online và số P có thể không là số nguyên tố.

Phân tích

Đây là một bài toán SRQ khá "thẳng", chỉ yêu cầu ta tính tích trên một đoạn bất kỳ. Ý tưởng "ngây thơ" nhất là duyệt qua mọi đoạn con được truy vấn để tìm tích của nó, mất $O(n^2)$. Ta cũng nghĩ đến việc áp dụng tích tiền tố đơn giản, tuy nhiên việc lấy ra một đoạn sẽ gặp khó khăn nếu P không phải là số nguyên tố. Các ý tưởng khác cho bài toán SRQ như cây phân đoạn và mảng thưa đều mất $O(\log n)$ cho mỗi truy vấn, khó qua được giới hạn của bài toán này. May mắn thay, sử dụng chia để trị vừa khít với bộ dữ liệu của bài toán.

Bằng cách sử dụng cách chia để trị như đã nói ở trên, ta dễ dàng khởi tạo mảng acc trong $O(n \log n)$. Khi trả lời một truy vấn $lq \ rq$, như đã nói ở trường hợp truy vấn offline, lq và rq phải thoả mãn $l \leq lq \leq mid$ và $mid + 1 \leq rq \leq r$ tại một vòng đệ quy.

Giả sử độ sâu lvl của một vòng đệ quy là số lần phải gọi đệ quy từ đoạn $[1, n]$, ta thấy hai đoạn có cùng độ sâu không có điểm chung. Do đó ta có thể lưu các giá trị acc đi kèm với độ sâu mà không sợ bị trùng lặp.





Quay lại với truy vấn $lq \ rq$, ta cần phải tìm một độ sâu sao cho lq và rq nằm về hai phía của mid của độ sâu này. Để giải quyết vấn đề này, ta gọi $mask[i]$ là một dãy bit, sao cho bit thứ j của dãy này bằng 0 nếu vị trí i nằm về bên trái mid (tính cả mid) ở độ sâu j , và 1 nếu vị trí này nằm về bên phải của mid (không tính mid). Ví dụ với dãy bằng 8 như trên hình, $mask[3] = (010)_2$, $mask[7] = (011)_2$. Như vậy, độ sâu thoả mãn lq và rq nằm về hai phía của mid ở độ sâu này là vị trí của bit đầu tiên bằng 1 từ phải qua trái trong dãy $mask[lq] \oplus mask[rq]$, với \oplus là phép xor.

Cài đặt

[► Nhấn để hiện code](#)

Chú ý thêm








- Kỹ thuật chia để trị có thể làm tối ưu khá tốt lời giải của một thuật toán. Đặc biệt trong các bài toán quy hoạch động, chia để trị là một trick tối ưu khá hiệu quả. Bạn đọc có thể tham khảo thêm tại [bài viết này](#) .
- Đối với các bài toán SRQ, còn một phương pháp hiệu quả vượt trội các phương pháp đã nói trên về độ phức tạp thời gian, đó là cấu trúc Sqrt Tree. Bạn đọc có thể tìm hiểu thêm về cấu trúc này tại [đây](#) .

Bài tập

Các bài chia để trị nói chung:

- [CERC 17 - I](#) 
- [VNOJ - NORMA](#) 
- [VNOJ - LIS2VN](#) 
- [UVA - Bit Maps](#) 
- [IOI 2011 - Race](#) 

Các bài toán SRQ:

- [VNOJ - XORSHIFT](#) 
- [Duyên hải Bắc Bộ 2023 - HKDATA](#)  (Bạn cần tham gia [contest](#)  để xem được đề bài)
- [DMOJ - Continued Fractions](#) 
- [USACO - Non-Decreasing Subsequences](#) 
- [Codeforces - Destiny](#) 
- [Codeforces - Timofey and our friends animals](#) 

Tài liệu tham khảo

- Steven Halim, Felix Halim (2013), *Competitive Programing 3*
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2022), *Introduction to Algorithms*, 4th edition
- Mark Berg , Otfried Cheong , Marc Kreveld , Mark Overmars (2008), *Computational Geometry - Algorithms and Applications*
- Wikipedia (Master's Theorem)
- [Bài giảng của Piotr Indyk tại MIT](#)  (bài NEAREST)
- USACO Guide Platinum, [Divide and Conquer - SRQ](#) 
- Codeforces, [Post của Wind_Eagle](#) 
- Codeforces, [Post của steveonalex](#) 

Được cung cấp bởi [Wiki.js](#)