Đáp án PREVNOI 2013-2014

Bài 1. ACM (Tác giả: Hồ Đắc Phương)

Thuật toán có thể suy ra từ cách làm thực tế. Nếu Tí làm tất cả $2n$ bài sẽ mất tổng thời gian là

$$S = \sum_{i=1}^{2n} a_i$$

Tuy nhiên Tí phải nhường lại n bài cho Tèo. Mỗi khi Tí nhường bài i cho Tèo, thời gian S sẽ giảm đi một lượng bằng $c_i = a_i - b_i$. Ta cần chọn n bài để Tí nhường cho Tèo sao cho thời gian S giảm đi nhiều nhất, tức là phải chọn n chỉ số i có c_i lớn nhất.

Vậy thuật toán đơn giản là sắp xếp các bài toán theo thứ tự tăng dần của các c_i, cho Tí làm n bài đầu và Tèo làm n bài sau. Chú ý là các hiệu c_i là các số nguyên $\in [-99 \dots 99]$ nên thuật toán sắp xếp ở đây có thể thay bằng phép đếm phân phối với độ phức tạp $\Theta(n)$

Mở rộng hơn, nếu các giá trị a_i, b_i là các số nguyên rất lớn hoặc số thực, vẫn có thể có thuật toán $O(n)$ bởi trên thực tế ta chỉ cần tìm trung vị của dãy $c[1 \dots 2n]$ mà không cần sắp xếp. Có nhiều thuật toán tuyến tính để tìm trung vị dãy, cách đơn giản nhất là cài đặt giải thuật tương tự như QuickSort, tuy nhiên sau khi chia đôi một đoạn ta chỉ quan tâm tới một trong hai đoạn con (chứ không phải sắp xếp cả hai đoạn con như QuickSort).

Phân loại: Ad-hoc, Sorting & Order Statistics.

Bài 2. PERIOD (Tác giả: Nguyễn Thế Hùng)

Xây dựng dãy $b_1, b_2, \dots, b_{2n-1}$ từ dãy A như sau:

$$\begin{aligned} b_1 &= a_1 + \Delta \\ b_2 &= a_2 + 2\Delta \\ &\dots \\ b_n &= a_n + n\Delta \\ b_{n+1} &= a_1 + (n+1)\Delta \\ b_{n+2} &= a_2 + (n+2)\Delta \\ &\dots \\ b_{2n-1} &= a_{n-1} + (2n-1)\Delta \end{aligned}$$

Có thể hiểu là b_i được tính bằng phần tử thứ i trong dãy A theo vòng tròn cộng thêm $i\Delta$.

Dễ thấy rằng nếu HUNGNT bắt đầu từ học sinh 1 thì thời gian giờ học kéo dài là $\max\{b[1 \dots n]\}$. Nếu bắt đầu từ học sinh 2 thì thời gian giờ học kéo dài là $\max\{b[2 \dots n+1]\} - \Delta$... Tổng quát: nếu bắt đầu từ học sinh i thì thời gian giờ học kéo dài là

$$\underbrace{\max\{b[i \dots i+n-1]\}}_{M_i} - (i-1)\Delta$$

Từ đây có thể tóm tắt thuật toán như sau: Với mỗi vị trí i, ta cần nhanh chóng xác định $M[i]$ là giá trị lớn nhất trong n phần tử tính từ b_i. Khi đó đáp số là $\min\limits_{i=1,2,\dots,n}\{M_i - (i-1)\Delta\}$.

Việc xác định giá trị lớn nhất trong các đoạn gồm n phần tử liên tiếp trong dãy B có thể thực hiện trong thời gian $O(n)$ bằng cách sử dụng hàng đợi hai đầu Q chứa các chỉ số trong B với các phép toán:

GetFront: Trả về phần tử đầu Q

GetRear: Trả về phần tử cuối Q

PopFront: Hủy phần tử đầu Q

PopRear: Hủy phần tử cuối Q

Push(i): Đẩy phần tử i vào cuối hàng đợi Q.

```
Q := ∅;
for i := 1 to 2n - 1 do
  begin
    while (Q ≠ ∅) & (a[GetRear] ≤ a[i]) do
      PopRear;
    Push(i);
    if GetFront + n ≤ i then
      PopFront;
    if i ≥ n then
      M[n - i + 1] := GetFront
  end;
return min_{i=1,2,...,n} {M_i - (i - 1)Δ};
```

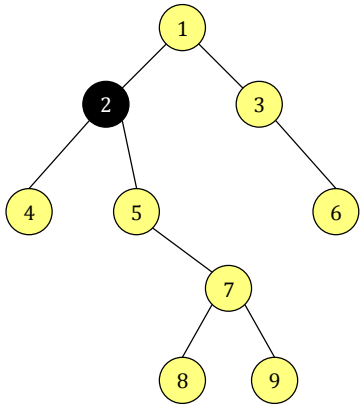
Các phép toán của hàng đợi hai đầu có thể cài đặt bằng mảng với hai chỉ số đầu cuối hoặc STL's deque. Tất cả các phép toán đó có độ phức tạp $O(1)$. Thuật toán sử dụng n lệnh Push vì vậy có tổng cộng không quá n lệnh PopRear và PopFront. Từ đó suy ra thời gian thực hiện giải thuật là $O(n)$.

Phân loại: Elementary data structures, double-ended queue, circular-list processing.

Bài 3. NETWORK (Tác giả: Nguyễn Tuấn Anh)

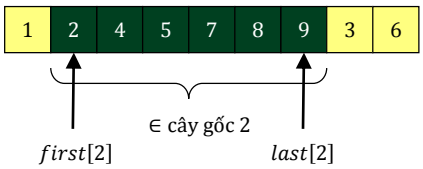
Duyệt cây bằng DFS:

- Mỗi khi duyệt tới đỉnh u đẩy ngay đỉnh đó vào cuối danh sách L, gọi $first[u]$ là vị trí của đỉnh u trong danh sách L.
- Mỗi khi duyệt xong đỉnh u, ghi nhận $last[u]$ là vị trí phần tử cuối danh sách L.




```
procedure Visit(u∈V);
begin
  first[u] := counter++;
  for ∀v:(u,v)∈E,first[v]=0 do
    Visit(v);
  last[u] := counter;
end;

begin
  counter := 0;
  first[1…n] := 0;
  Visit(1);
end;
```



Với cách xây dựng danh sách L như vậy, với mỗi đỉnh u thì các đỉnh nằm trong cây con gốc u sẽ được liệt kê trong một đoạn liên tiếp từ vị trí $first[u]$ tới $last[u]$

Sắp xếp các cạnh tăng dần theo trọng số w, sắp xếp các truy vấn tăng dần theo trọng số truy vấn c. Xử lý riêng biệt 2 loại truy vấn.

Đối với các truy vấn loại $P(a, b, c)$ xét theo thứ tự tăng dần của c

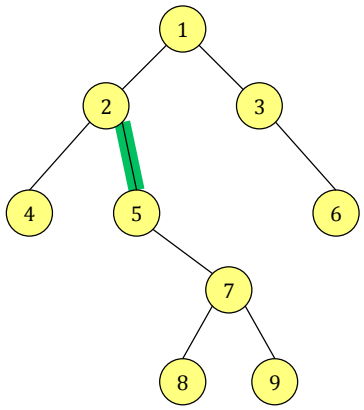
Khởi tạo: Các cạnh đều là "nhạt".

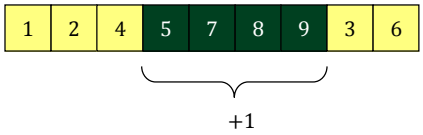
Dựa vào danh sách cạnh đã sắp xếp, tô đậm những cạnh có trọng số $\leq c$ nếu chúng chưa được tô. Ta cần là đếm số cạnh đậm trên đường đi từ a tới b.

$\forall v \in V$, gọi $d[v]$ là số cạnh đậm trên đường đi từ gốc 1 tới v. Khi đó câu trả lời cho truy vấn $P(a, b, c)$ sẽ là

$$d(a) + d(b) - 2d(p)$$

với p là tiền bối chung thấp nhất của a và b. Đường đi từ a tới b sẽ xuất phát từ a đi lên p rồi xuống b.





Vấn đề còn lại là cần có cơ chế cập nhật và truy vấn các giá trị $d[v]$ một cách hợp lý khi có thêm các cạnh được tô đậm. Để ý rằng nếu u là cha của v thì khi tô đậm cạnh (u, v), các nhãn $d[.]$ của các đỉnh thuộc cây gốc v sẽ bị tăng lên 1, các đỉnh này theo giải thích ở trên, nằm trong một đoạn liên tiếp trong danh sách L từ vị trí $first[v]$ tới vị trí $last[v]$.

Bằng việc cài đặt một cấu trúc dữ liệu quản lý phạm vi như BST, Segment trees, Fenwick trees. Ta có thể thực hiện cả hai loại phép toán sau với thời gian $O(\log n)$ mỗi phép toán.

- Tăng nhãn $d[.]$ của một đoạn liên tiếp các đỉnh trong danh sách L lên 1 đơn vị
- Truy vấn nhãn $d[.]$ của một phần tử trong danh sách L

(Cập nhật 1 đoạn, truy vấn 1 điểm)

Trong chương trình mẫu chúng tôi sử dụng Fenwick trees. Việc tô đậm 1 cạnh mất thời gian $O(\log n)$, việc truy vấn nhãn $d[.]$ của một đỉnh mất thời gian $O(\log n)$, việc tìm LCA (tiền bối chung thấp nhất) của hai đỉnh mất thời gian $O(\log n)$ hoặc $O(1)$ tùy theo thuật toán LCA được sử dụng.

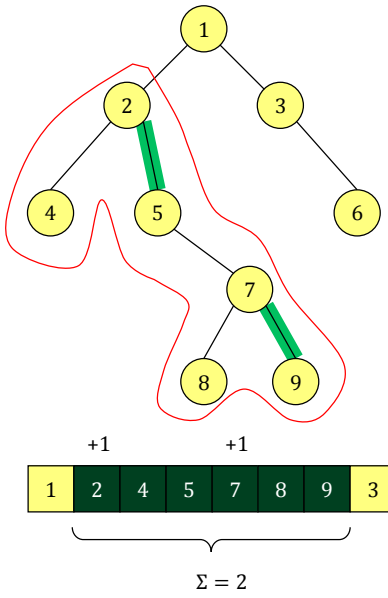
Đối với các truy vấn loại $P(i, c)$ xét theo thứ tự tăng dần của c.

Khởi tạo: Các cạnh đều là "nhạt".

Dựa vào danh sách cạnh đã sắp xếp, tô đậm những cạnh có trọng số $\leq c$ nếu chúng chưa được tô. $\forall v \in V$, gọi $q[v]$ là số cạnh đậm nối từ v tới con của nó.

Xét cạnh thứ i là (a, b),

- Trường hợp a là cha của b thì câu trả lời cho truy vấn là số cạnh đậm trong cây gốc b.
- Ngược lại nếu b là cha của a thì câu trả lời cho truy vấn là tổng số cạnh đậm trừ 1 trừ đi tiếp số cạnh đậm trong cây gốc a.



Vấn đề còn lại là cần có cơ chế cập nhật các nhãn $q[.]$ và truy vấn tổng các nhãn $q[.]$ trong một cây con khi có thêm các cạnh được tô đậm. Nếu u là cha của v thì việc tô đậm cạnh (u, v) sẽ làm $q[u]$ tăng lên 1. Ngoài ra việc tính tổng các nhãn $q[.]$ trong một cây con thì cũng tương đương với việc tính tổng các nhãn $q[.]$ trong một đoạn liên tiếp của danh sách L như đã giải thích về quá trình DFS.

Bằng việc cài đặt một cấu trúc dữ liệu quản lý phạm vi như BST, Segment trees, Fenwick trees. Ta có thể thực hiện cả hai loại phép toán sau với thời gian $O(\log n)$ mỗi phép toán.

- Tăng nhãn $q[.]$ của một phần tử trong danh sách L lên 1 đơn vị
- Truy vấn tổng các nhãn $q[.]$ của một loạt các phần tử liên tiếp trong danh sách L

(Cập nhật 1 điểm, truy vấn 1 đoạn)

Trong chương trình mẫu chúng tôi sử dụng Fenwick trees. Việc tăng nhãn $q[.]$ của một phần tử mất thời gian $O(\log n)$, việc truy vấn tổng các nhãn $q[.]$ trong một đoạn liên tiếp mất thời gian $O(\log n)$.

Thời gian thực hiện giải thuật: $O((n + m) \log n)$

Loại giải thuật: Sorting, Range-queries, LCA, Graph, DFS.

Bài 4. BEADS (Tác giả: Đỗ Đức Đông)

Với dãy kích thước vỏ ốc $A = (a_1, a_2, \dots, a_n)$, ta nhận thấy rằng nếu chọn vỏ ốc i làm vỏ ốc đầu tiên thì:

- i và các vỏ ốc nối tiếp về bên phải chuỗi hạt phải lập thành một dãy tăng
- i và các vỏ ốc nối tiếp về bên trái chuỗi hạt phải lập thành một dãy giảm.

Hai dãy này đều là dãy con của $(a_i, \dots, a_n)$ và dĩ nhiên chỉ có 1 phần tử chung duy nhất là a_i

Từ đó suy ra thuật toán: Với mỗi vị trí i, ta tìm $f[i]$ là độ dài dãy con tăng dài nhất bắt đầu từ vị trí i và $g[i]$ là độ dài dãy con giảm dài nhất bắt đầu từ vị trí i. Việc tìm các $f[1 \dots n]$ và $g[1 \dots n]$ là bài toán quy hoạch động cơ bản với độ phức tạp $O(n \log m)$ với m là độ dài dãy con tăng/giảm dài nhất.

Đáp số là:

$$\max_{i=1,2,\dots,n} \{f[i] + g[i] - 1\}$$

Loại giải thuật: Dynamic programming.

Bài 5. QUEEN (Tác giả: Lê Đôn Khuê)

Bao quanh viền ngoài bàn cờ bởi các chướng ngại vật (cầm canh)

Xét từng hàng của bàn cờ, với mỗi hàng, đếm số ô trống giữa hai chướng ngại vật liên tiếp, đó chính là số ô mà mỗi quân hậu tại ô trống có thể khống chế theo chiều ngang.

Tương tự như vậy với từng cột, từng đường chéo.

Sau 4 lần quét với từng hàng, cột, đường chéo chính, đường chéo phụ. Cuối cùng, với mỗi ô trống, tính tổng số ô mà nó không chế theo 4 hướng và trừ đi 3.

Chú ý là với mỗi ô (x, y), số ô bị ô (x, y) không chế = số ô không chế ô (x, y).

Để cài đặt ngắn gọn, ta có thể sử dụng vài kỹ thuật nhỏ:

Số ô trống giữa hai chướng ngại vật liên tiếp (x_1, y_1) và (x_2, y_2) bất kể đang quét theo hướng nào có thể tính bằng công thức

$$\max\{|x_2 - x_1|, |y_2 - y_1|\}$$

Việc quét một hàng/cột/đường chéo có thể thực hiện bởi hai tham số: Ô đầu tiên và vector chỉ hướng quét

Độ phức tạp $O(mn)$

Một cách khác là dùng quy hoạch động theo $f[i, j, k]$ là số ô mà quân hậu ở ô (i, j) ăn được theo hướng k ($k = 1, \dots, 8$)

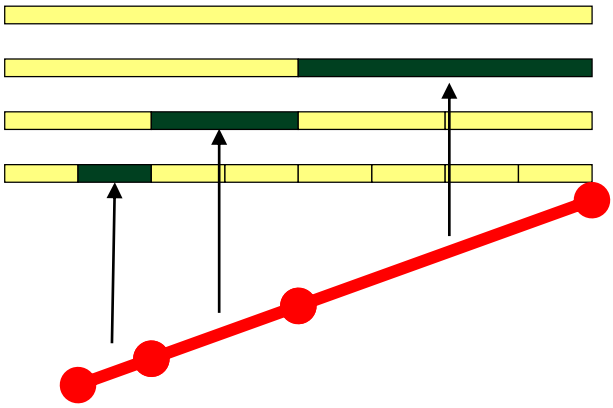
Loại giải thuật: Ad-hoc

Bài 6. JEWEL (tác giả: Nguyễn Duy Khương)

Mỗi mặt hàng (s, e, v, d) (bán với giá khởi điểm v từ s tới e, đi mỗi km tăng d đồng) có thể biểu diễn bởi một đoạn thẳng nối hai điểm AB với $A = (s, v)$ và $B = (e, v + (e - s) * d)$. Mỗi điểm (x, y) trên đoạn thẳng cho biết giá bán mặt hàng tại x là y đồng.

Sử dụng một segment trees T quản lý n điểm bán hàng trên đoạn đường.

- Nút gốc 1 quản lý các điểm $1 \dots n$
- Nếu nút i quản lý các điểm $L_i \dots H_i$ thì con trái $2i$ quản lý từ điểm $L_i..M$ và con phải $2i + 1$ quản lý các điểm $M + 1 \dots H_i$. Ở đây $M = [(L_i + H_i)/2]$
- Mỗi nút chứa một đoạn thẳng có hoành độ đúng bằng phạm vi nó quản lý.



Với mỗi đoạn thẳng AB, ta tách nó ra thành $O(\log n)$ đoạn thẳng và chèn vào cây theo thuật toán tương tự như truy vấn khoảng:

- Việc chèn sẽ được thực hiện từ nút gốc
- Xét phép chèn vào nút i:
  - Nếu $[L_i, H_i] \cap [A.x, B.x] = \emptyset$, bỏ qua
  - Nếu $[L_i, H_i] \subseteq [A.x, B.x]$, cập nhật nút i bởi phần đoạn thẳng hạn chế trên hoành độ $[L_i, H_i]$, thuật toán cập nhật sẽ nói sau.
  - Trường hợp khác, chèn đoạn thẳng vào 2 nút con bằng đệ quy.

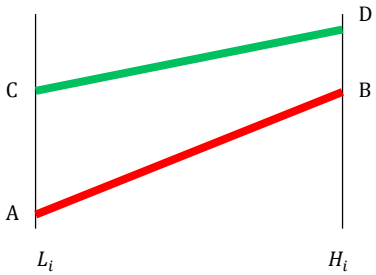
Hình bên là mô tả một đoạn thẳng bị tách làm 3 khúc đưa vào cập nhật 3 nút khác nhau của cây.

Tiếp theo ta trình bày thuật toán cập nhật nút i bởi đoạn thẳng AB

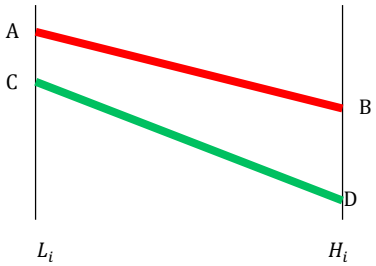
Bởi mỗi nút trên cây chỉ chứa được một đoạn thẳng nên nếu trong nút i chưa có đoạn thẳng nào thì ta đưa luôn đoạn AB vào nút i.

Nếu trong nút i đã có sẵn đoạn thẳng CD:

TH1: Đoạn AB nằm dưới CD. Có nghĩa là: giá mặt hàng mới biểu diễn bởi AB xét trên khoảng $[L_i, H_i]$ không cao hơn giá mặt hàng cũ biểu diễn bởi CD đang chứa trong nút. Vì ta chỉ quan tâm tới giá cao, đoạn AB bị bỏ qua.



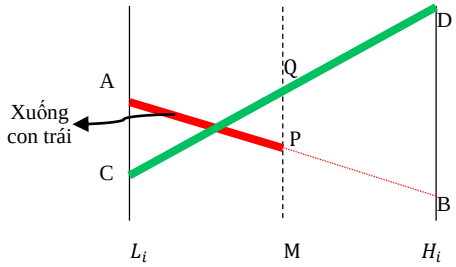
TH2: Đoạn AB nằm trên CD. Có nghĩa là: giá mặt hàng mới biểu diễn bởi AB xét trên khoảng $[L_i, H_i]$ cao hơn hoàn toàn giá mặt hàng cũ biểu diễn bởi CD. Ta thay đoạn CD trong nút bởi AB.

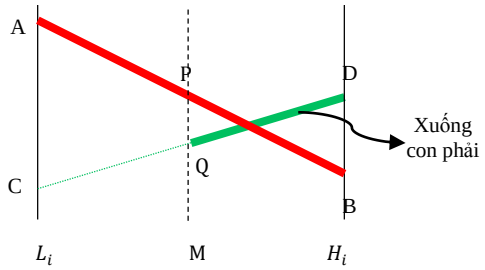


TH3: Hai đoạn AB và CD cắt nhau, không giảm tính tổng quát, giả sử A cao hơn C và B thấp hơn D. Xét giá tại điểm $M = [(L_i + H_i)/2]$ ứng với điểm $P \in AB$ và $Q \in CD$.

Nếu P thấp hơn Q, lưu lại đoạn CD trong nút i, đưa đoạn AP xuống cập nhật nút con trái $2i$ bằng đệ quy, đoạn PB giờ đây không còn ý nghĩa gì nữa. (Hình trái)

Nếu P cao hơn Q, lưu lại đoạn AB trong nút i, đưa đoạn QD xuống cập nhật nút con phải $2i + 1$ bằng đệ quy, đoạn CQ giờ đây không còn ý nghĩa gì nữa. (Hình phải)





Việc cập nhật nút tương đương với một phép di chuyển từ nút đó xuống phía lá có thời gian thực hiện $O(\log n)$.

Tóm tắt: Mỗi mặt hàng được tách ra thành $O(\log n)$ khúc, mỗi khúc được đưa vào cập nhật 1 nút, việc cập nhật 1 nút lại có thể dẫn tới $O(\log n)$ quá trình chia đôi đoạn thẳng mang xuống cập nhật nút con…

Toàn bộ công đoạn xử lý n mặt hàng: $O(n \times \log^2 n)$. Trên thực tế rất khó làm test để việc cập nhật nào cũng rơi vào trường hợp 3 nhất là khi ta dùng một heuristic trộn các đoạn thẳng lên và xét theo thứ tự ngẫu nhiên.

Việc còn lại khá đơn giản: Với mỗi vị trí x, ta mất thời gian $O(\log n)$ để xét nút lá quản lý trực tiếp x và các nút tiền bối của nó trên cây T, xem ở nút nào mặt hàng (biểu diễn bởi đoạn thẳng trong nút) có giá cao nhất.

Kết luận: Thời gian thực hiện giải thuật $O(n \times \log^2 n)$

Loại giải thuật: Advanced data structures, Computational geometry.

Nói thêm: Có thuật toán cùng độ phức tạp dựa trên convexhull nhưng tôi không cài nổi.

http://en.wikipedia.org/wiki/Dynamic_convex_hull