

Interval Tree trên tập đoạn thẳng

Interval Tree trên tập đoạn thẳng

Tác giả: Vũ chipchip Phúc Hoàng

Bài toán

Cho một tập hợp chứa các đường thẳng có dạng $ax + b$, mỗi đường thẳng được biểu diễn bằng một cặp số (a, b) . Cần thực hiện hai loại thao tác:

1. Thêm một đường thẳng vào tập hợp.
2. Trả lời xem tại hoành độ q , điểm nào thuộc ít nhất một đường thẳng trong tập có tung độ lớn nhất. Nói cách khác, đường thẳng (a, b) nào có $aq + b$ lớn nhất.

Để giải bài toán này, hai cách phổ biến là ứng dụng [bao lồi](#) và sử dụng cây Interval Tree lưu đoạn thẳng. Sau đây là những ưu điểm và nhược điểm của IT đoạn thẳng so với bao lồi.

Ưu điểm

1. Ứng dụng được với đoạn thẳng chứ không chỉ đường thẳng. Đây là ưu điểm lớn nhất của IT đoạn thẳng so với bao lồi, khi tập hợp cần xử lý là tập đoạn thẳng chứ không phải đường thẳng (tức là đường thẳng $ax + b$ chỉ tồn tại khi x thuộc một khoảng (l, h) nhất định), bao lồi sẽ không thể làm được.
2. Thực hiện thao tác thêm đường thẳng (đoạn thẳng) một cách dễ dàng. Bao lồi gặp nhược điểm lớn khi thêm đường thẳng mà hệ số góc a không tăng dần hoặc giảm dần. Mặc dù không phải là không thể làm được, nhưng bao lồi khi đó phải biểu diễn bằng cấu trúc khác không phải stack, gây khó khăn lớn khi code.
3. Dễ code. Chính vì hai ưu điểm ở trên, IT đoạn thẳng rất tổng quát và không cần phải xét trường hợp phụ thuộc vào bài toán như bao lồi. Đa số các bài toán, phần [Update](#) và [Query](#) của IT đoạn thẳng gần như giống hệt nhau. Phần thân chương trình cũng rất ngắn gọn.

Nhược điểm

1. Phụ thuộc vào kích thước hoành độ x . Vì IT đoạn thẳng xử lý trên khoảng của hoành độ, với bài toán mà query x lớn hoặc x không phải số nguyên không thể biểu diễn bằng IT bình thường. Có thể thay thế bằng rời rạc hóa các tọa độ hoặc IT động, nhưng so với bao lồi đây là một nhược điểm đáng kể khi bao lồi hoàn toàn không phụ thuộc vào x .
2. Bộ nhớ và thời gian lớn. Lưu một cây IT chứa hai số nguyên a, b tốn bộ nhớ hơn nhiều so với stack bao lồi. Xử lý trên cây IT cũng chậm hơn chặt nhệ phân trên bao lồi. Về độ phức tạp, có thể so sánh qua bảng sau

Điều kiện bài toán	IT đoạn thẳng		Bao lồi	
	Update	Query	Update	Query
Xử lý trên tập đường thẳng	$O(\log(X))$	$O(\log(X))$	$O(1)$	$O(\log(n))$
Xử lý trên tập đoạn thẳng	$O(\log^2(X))$	$O(\log(X))$	//	//
Xử lý trên tập đường thẳng, truy vấn tăng dần hoặc giảm dần	$O(\log(X))$	$O(\log(X))$	$O(1)$	$O(1)$

Lưu ý: Ở đây ta giả sử các đường thẳng thêm vào có hệ số a tăng dần hoặc giảm dần, bao lồi được biểu diễn bằng stack.

Tóm lại, so với cách ứng dụng bao lồi, sử dụng IT đoạn thẳng là một phương pháp tổng quát hơn nhưng chậm và tốn nhiều bộ nhớ hơn. Sau đây là những phân tích cơ bản về thuật toán.

Ý tưởng

Xây dựng một cây Interval Tree để quản lý tập các đoạn thẳng, mỗi nút của cây quản lý một khoảng trên trục hoành. Thông tin lưu ở mỗi nút trên cây sẽ là đoạn thẳng đặc trưng cho khoảng nó quản lý. Đoạn thẳng này phải phủ kín khoảng, tức là đoạn $ax + b$ có khoảng x bao lấy khoảng do nút quản lý (nếu là đường thẳng thì luôn phủ kín khoảng do nút quản lý). Đoạn thẳng được lưu trong nút phải cao hơn tất cả các đoạn khác tại một vị trí nào đó thuộc khoảng (nếu không thì không cần quan tâm đến đoạn đó). Ý nghĩa của việc lưu này là với một query q bất kỳ, đoạn $aq + b$ cao nhất sẽ được lưu trong một nút nào đó của cây IT quản lý khoảng chứa q . Cách lưu đoạn thẳng này khá trừu tượng, nếu bạn đọc phần này chưa hiểu, nên bỏ qua để xem cách [Query](#) và [Update](#) trên cây rồi đọc lại phần này sau.

Như vậy, thông tin lưu trên cây IT sẽ được biểu diễn bằng một mảng line, line là một cặp số (a, b) biểu diễn đường thẳng.

```
1 | line it[MAXX * 4]; // MAXX là giới hạn trục hoành
```

Ngoài ra, có thể thêm một vài mảng phụ cần thiết cho IT như [low](#) , [high](#) , [leaf](#) , ...

Ta định nghĩa hàm `Get(line d, int x)` cho biết tung độ của điểm thuộc đường thẳng d tại hoành độ x .

```
1 | int Get(line d, int x)
2 | {
3 |     return d.a * x + d.b;
4 | }
```

Query

Ta sẽ trả lời cho query q , xem tại hoành độ $x = q$, tìm tung độ cao nhất của một điểm thuộc một đoạn trong tập. Như đã nói ở trên, IT lưu các đoạn thẳng đảm bảo trong các nút cây quản lý khoảng chứa q có một nút lưu

đoạn thẳng đạt tung độ cao nhất (làm thế nào để được như vậy thì xem phần Update). Vậy ở đây, muốn trả lời cho query q , ta đi từ gốc xuống nút lá quản lí điểm q , trên đường đi update đáp số bằng tung độ cao nhất tại điểm q của đoạn thẳng do nút đó quản lí.

```

1  int Query(int node, int pos)
2  {
3      if(low[node] > pos || high[node] < pos)
4      {
5          return -oo;
6      }
7      res = Get(it[node], pos);
8      if(low[node] == high[node])
9      {
10         return res;
11     }
12     res = max(res, Query(node * 2, pos));
13     res = max(res, Query(node * 2 + 1, pos));
14     return res;
15 }

```

Độ phức tạp: $O(\log(MAXX))$

Update

Thêm một đoạn thẳng vào tập hợp, ta phải thay đổi những nút trên cây IT quản lí khoảng ứng với đoạn thẳng đó. Việc đầu tiên, giống như Update trên cây IT cơ bản, ta phải chia đoạn cần Update ra thành những khoảng IT.

```

1  void Update(int node, int l, int h, line val)
2  {
3      if(low[node] > h || high[node] < l)
4      {
5          return;
6      }
7      if(low[node] >= l && high[node] <= h)
8      {
9          // Do something
10         return;
11     }
12     Update(node * 2, l, h, val);
13     Update(node * 2 + 1, l, h, val);
14 }

```

Độ phức tạp của phần chia khoảng này là $O(\log(MAXX))$, giống như IT cơ bản. Nếu đoạn cần Update là đường thẳng ($l = low[1], h = high[1]$) thì không mất thời gian chia khoảng, độ phức tạp chỉ là $O(1)$.

Bây giờ việc phải làm là điền vào chỗ `// Do Something`. Ta có một đường thẳng `val` và đường thẳng `it[node]`, cả hai đều chỉ được xét trong khoảng từ `low[node]` đến `high[node]`. Lấy `mid` là điểm giữa của khoảng $(mid = (low[node] + high[node]) / 2)$. Ta sẽ thay đổi nút `it[node]` và cả các con của nó. Có 6 trường hợp có thể xảy ra:

1. `it[node]` hoàn toàn nằm trên `val`. Trường hợp này ta chỉ bỏ qua mà không làm gì, vì `val` chắc chắn không bao giờ đạt max trong khoảng `low[node]` đến `high[node]`.

```

1 | if(Get(it[node], low[node]) >= Get(val, low[node]) && Get(it[node], high[node]
2 | {
3 |     return;
4 | }
```

2. `it[node]` hoàn toàn nằm dưới `val`. Trường hợp này ta gán `it[node]` bằng `val`, `it[node]` cũ không còn giá trị khi tìm max.

```

1 | if(Get(it[node], low[node]) <= Get(val, low[node]) && Get(it[node], high[node]
2 | {
3 |     it[node] = val;
4 |     return;
5 | }
```

3. Nửa bên trái của `it[node]` hoàn toàn nằm trên nửa bên trái của `val`. Vậy `val` chắc chắn không bao giờ đạt max tại nửa trái của khoảng `node`, ta giữ lại `it[node]` tại `node` và down `val` xuống con phải $(node * 2 + 1)$.

```

1 | if(Get(it[node], low[node]) >= Get(val, low[node]) && Get(it[node], mid) >= Ge
2 | {
3 |     Update(node * 2 + 1, l, h, val);
4 |     return;
5 | }
```

4. Nửa bên trái của `it[node]` hoàn toàn nằm dưới nửa bên trái của `val`. Tương tự như trên, ta down `it[node]` xuống con phải của node và update `it[node]` bằng `val`.

```

    if(Get(it[node], low[node]) <= Get(val, low[node]) && Get(it[node], mid) <= Ge
    {
```

```

3 | Update(node * 2 + 1, l, h, it[node]);
4 | it[node] = val;
5 | return;
6 | }

```

5. Nửa bên phải của `it[node]` hoàn toàn nằm trên nửa bên phải của `val`.

```

1 | if(Get(it[node], mid + 1) >= Get(val, mid + 1) && Get(it[node], high[node]) >=
2 | {
3 |     Update(node * 2, l, h, val);
4 |     return;
5 | }

```

6. Nửa bên phải của `it[node]` hoàn toàn nằm dưới nửa bên phải của `val`.

```

1 | if(Get(it[node], mid + 1) <= Get(val, mid + 1) && Get(it[node], high[node]) <=
2 | {
3 |     Update(node * 2, l, h, it[node]);
4 |     it[node] = val;
5 |     return;
6 | }

```

Sau khi xét xong 6 trường hợp ở trên, ta đã xử lí xong việc Update đoạn `val` trong một khoảng `low[node]`, `high[node]`. Độ phức tạp của thao tác này là $O(\log(MAXX))$, vì có thể phải đi từ `node` cho đến lá. Có thể thấy, cây IT có đầy đủ thông tin về đoạn thẳng đạt max tại một hoành độ nhất định, vì ta chỉ loại những đoạn thẳng mà hoàn toàn không còn giá trị (trường hợp 1 và trường hợp 2), còn những đoạn thẳng vẫn có thể đạt max tại một vị trí nào đấy luôn được bảo tồn.

Độ phức tạp: $O(\log^2(MAXX))$. $O(\log(MAXX))$ khi chia khoảng, $O(\log(MAXX))$ khi update trên một khoảng. Nếu update đường thẳng thì không mất thời gian chia khoảng, độ phức tạp tổng cộng là $O(\log(MAXX))$.

Mở rộng

Query và Update ở trên là những thao tác cơ bản nhất của IT đoạn thẳng. Ngoài ra, có thể có thêm nhiều thông tin phụ đính kèm với đoạn thẳng, tùy thuộc vào đề bài toán.

Có nhiều cách để biểu diễn đoạn thẳng trong cây IT ngoài $ax + b$. Ví dụ, có thể biểu diễn đoạn thẳng bằng cách lưu tọa độ 2 điểm đầu mút của đoạn. Tùy vào đề bài toán mà có cách biểu diễn hợp lí nhất.

Ứng dụng

Bài toán tìm max, min của $ax + b$ thường đi kèm với thuật toán quy hoạch động, chẳng hạn như bài toán quy hoạch động có công thức $f[i] = \max(a[j] \times x[i] + b[j] + c)$, ta cần tìm $j < i$ sao cho hàm đó đạt max. Bao lồi cũng là phương pháp thường được sử dụng trong bài toán này. Hạn chế của bao lồi là $a[j]$ phải tăng dần hoặc giảm dần (nếu không sẽ phải sử dụng cấu trúc khác stack để biểu diễn bao lồi, code rất khó khăn). Hạn chế của IT đoạn thẳng là $x[i]$ phải nguyên và nhỏ để có thể biểu diễn trên IT (nếu không sẽ phải sử dụng IT động hoặc rời rạc hóa).

Ngoài ra, có một số bài toán yêu cầu tìm max, min trên tập đoạn thẳng. Đây là những bài toán IT đoạn thẳng gần như là cách làm duy nhất.

Một số câu hỏi:

Để hiểu rõ về IT đoạn thẳng, bạn hãy tự trả lời một số câu hỏi sau:

1. Trong trường hợp nào thì một nút không có thông tin gì cả?
2. Trong các trường hợp 4 và 6 của phần [Update](#), tại sao phải gán lại `val` cho `it[node]` ?
3. Giả sử thay vì truy vấn theo điểm, ta truy vấn theo khoảng, tức là trả lời xem tại tất cả các điểm trong một khoảng nào đó, đoạn thẳng nào đạt chiều cao lớn nhất / nhỏ nhất. Giả sử khoảng này nằm hoàn toàn trong phạm vi quản lý của một nút nào đó, liệu ta có thể trả luôn kết quả là đoạn thẳng lưu trong nút đó không? Vì sao?

Bài tập áp dụng

Một số bài tập "quy hoạch động bao lồi" truyền thống


- ▶ [VNOJ - VMPIZZA](#) 
- ▶ [CF 189 - Div 1 - C](#) 
- ▶ [SPOJ - ACQUIRE](#) 
- ▶ [SPOJ - APIO10A](#) 

Để làm những bài tập này, đầu tiên ta sẽ giải bằng cách quy hoạch động với độ phức tạp $O(N^2)$. Công thức quy hoạch động sẽ có dạng là $f[i] = \max / \min(a[j] \times x[i] + b[j] + c)$, với mọi j từ 1 đến $i - 1$. Để giảm độ phức tạp xuống $O(N \log N)$, ta sẽ sử dụng bao lồi hoặc IT đoạn thẳng. Lưu ý là với cách bao lồi, stack bao lồi phải đảm bảo $a[j]$ tăng dần hoặc giảm dần, nếu không phải lọc ra sao cho tính chất này thỏa mãn. Lưu ý rằng bao lồi chỉ có thể làm được khi hệ số góc tăng dần hoặc giảm dần.

USACO - Fencing the Herd

Bài này yêu cầu tìm $(Ax + By)$ max và min khi cho điểm (x, y) bất kì, hay là $(Ax/y + B)$ max và min.

Đây chính là dạng chuẩn của bài toán bao lồi và IT đoạn thẳng. Tuy nhiên làm bao lồi trong trường hợp này cực kì khó khăn, vì hệ số góc A không đảm bảo tăng dần hoặc giảm dần. Để có thể làm bao lồi với bài này, ta phải sử dụng cấu trúc dữ liệu lưu bao lồi sao cho hệ số góc A vẫn tăng hoặc giảm, cách đơn giản nhất là trong quá

trình thêm (A, B) ta sử dụng một buffer có sức chứa là \sqrt{Q} , khi nào buffer đầy thì gộp vào bao lồi. Lúc query thì tìm max, min trên cả bao lồi và buffer. Solution bao lồi chi tiết xem [ở đây](#) .

Còn với IT đoạn thẳng, ta cũng gặp khó khăn vì query không phải là số nguyên, và x/y cũng rất lớn. Tuy nhiên ta có thể xử lý offline đơn giản bằng cách đọc hết tất cả các query, lưu lại các điểm (x/y) , rồi rạc hóa lại, và xây dựng cây IT đoạn thẳng trên tập điểm đã rời rạc hóa đấy. Trong bài này, cách IT đoạn thẳng đơn giản hơn nhiều so với cách bao lồi.

VNOJ - VOMARIO

Bài "độc quyền" của IT đoạn thẳng. Trong bài này, ta cũng tìm công thức quy hoạch động $O(N^2)$: $f[i] = \max(a[j] \times x[i] + b[j] + c)$.

Tuy nhiên, đáng lưu ý là mỗi cặp $(a[j], b[j])$ chỉ được tính trong một khoảng $x[i]$ nào đó, còn $x[i]$ nằm ngoài khoảng đó thì cặp $(a[j], b[j])$ này không được phép chọn để lấy max. Đây chính là tính chất "đoạn thẳng" thay vì "đường thẳng". Bài này không thể sử dụng bao lồi để giải được.

Bài khác

- [VNOJ - JEWELNB](#) 
- [SPOJ - PTIT133B](#) 

Được cung cấp bởi [Wiki.js](#)