

Segment Tree

Lazy propagation

Lorenzo Ferrari, Davide Bartoli

February 22, 2023

Table of contents

Range update – point query

Lazy propagation

Problemi

Range update – point query

Problema

Dato un array di $N \leq 200'000$ elementi, supporta le seguenti operazioni:

- ▶ aumenta di x gli elementi con indice in $[l, r]$
- ▶ trova quanto vale l' i -esimo elemento

<https://cses.fi/problemset/task/1651>

Range update – point query

Fin'ora abbiamo affrontato update su un punto e query su un range: qui la situazione è invertita.

Range update – point query

Fin'ora abbiamo affrontato update su un punto e query su un range: qui la situazione è invertita.

Esistono più modi per affrontare il problema, il più semplice si basa sulla seguente osservazione

Osservazione

Ogni query su un range, tocca solo i segmenti toccati dai singoli point-update nel range.

Range update – point query

Fin'ora abbiamo affrontato update su un punto e query su un range: qui la situazione è invertita.

Esistono più modi per affrontare il problema, il più semplice si basa sulla seguente osservazione

Osservazione

Ogni query su un range, tocca solo i segmenti toccati dai singoli point-update nel range.

- possiamo sfruttare questa proprietà per invertire le operazioni

Range update – point query

Fin'ora abbiamo affrontato update su un punto e query su un range: qui la situazione è invertita.

Esistono più modi per affrontare il problema, il più semplice si basa sulla seguente osservazione


Osservazione

Ogni query su un range, tocca solo i segmenti toccati dai singoli point-update nel range.

- possiamo sfruttare questa proprietà per invertire le operazioni

La funzione update assomiglia alla precedente funzione query, la funzione query assomiglia alla precedente funzione update.

Query iterativa




```
long long query(int p) {  
    int ans = 0;  
    for (p += n; p > 0; p >= 1) {  
        ans += t[p];  
    }  
    return ans;  
}
```


Query ricorsiva



```
long long query(int i, int tl, int tr, int p) {  
    if (p < tl || tr < p) return 0;  
    if (tl == tr) {  
        return t[i];  
    } else {  
        int tm = (tl + tr) / 2;  
        if (p <= tm) {  
            return t[i] + query(2*i, tl, tm, p);  
        } else {  
            return t[i] + query(2*i+1, tm+1, tr, p);  
        }  
    }  
}
```



```
void update(int i, int tl, int tr, int l, int r, long long x) {
    if (r < tl || tr < l) return;
    if (l <= tl && tr <= r) t[i] += x;
    else {
        int tm = (tl + tr) / 2;
        update(2*i, tl, tm, l, r, x);
        update(2*i+1, tm+1, tr, l, r, x);
    }
}

void update(int l, int r, long long x) {
    update(1, 0, n-1, l, r, x);
}
```

Lazy propagation

Problema

Lazy propagation

Dato un array di N numeri, rispondi alle seguenti query:

- ▶ aggiungi k a tutti gli elementi dell'intervallo $[a, b]$
- ▶ calcola la somma massima di un sottoarray di un intervallo $[l, r]$

Il problema ricorda quello visto nella prima lezione sul segment tree, ma questa volta dobbiamo aggiornare più elementi contemporaneamente.

Lazy propagation

Idea

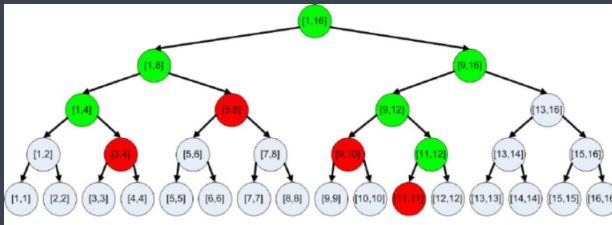
Potremmo chiamare l'algoritmo visto nella prima lezione $b - a + 1$ volte per ogni update, ma ovviamente questo sarebbe troppo lento.

Lazy propagation

Idea

Potremmo chiamare l'algoritmo visto nella prima lezione $b - a + 1$ volte per ogni update, ma ovviamente questo sarebbe troppo lento. Ora l'update è su un range come era la query, quindi possiamo riutilizzare la stessa idea:

- facciamo una dfs per scomporre l'intervallo in un insieme di nodi che formano il nostro intervallo $[a, b]$



Lazy propagation

Idea

Come facciamo ad aggiornare questi nodi dell'albero?

Lazy propagation

Idea

Come facciamo ad aggiornare questi nodi dell'albero?

Possiamo tenerci un nuovo campo `aggiungi` che tiene traccia di quanto abbiamo aggiunto agli elementi che sono sotto questo nodo.

Lazy propagation

Idea

Come facciamo ad aggiornare questi nodi dell'albero?

Possiamo tenerci un nuovo campo `aggiungi` che tiene traccia di quanto abbiamo aggiunto agli elementi che sono sotto questo nodo.

In questo modo possiamo semplicemente aggiornare il campo `aggiungi` e fermarci.

Lazy propagation

Idea

Come facciamo ad aggiornare questi nodi dell'albero?

Possiamo tenerci un nuovo campo `aggiungi` che tiene traccia di quanto abbiamo aggiunto agli elementi che sono sotto questo nodo.

In questo modo possiamo semplicemente aggiornare il campo `aggiungi` e fermarci.

Ora però le nostre informazioni sono "sparse" tra i nodi dell'albero, e dobbiamo capire come recuperarle per rispondere alle query.

Lazy propagation

Idea

Come facciamo ad aggiornare questi nodi dell'albero?

Possiamo tenerci un nuovo campo `aggiungi` che tiene traccia di quanto abbiamo aggiunto agli elementi che sono sotto questo nodo.

In questo modo possiamo semplicemente aggiornare il campo `aggiungi` e fermarci.

Ora però le nostre informazioni sono "sparse" tra i nodi dell'albero, e dobbiamo capire come recuperarle per rispondere alle query.

L'idea è che ogni volta che visitiamo un nodo, propaghiamo le informazioni di aggiunta verso i nodi figli.

Lazy propagation

Idea

Come facciamo ad aggiornare questi nodi dell'albero?

Possiamo tenerci un nuovo campo `aggiungi` che tiene traccia di quanto abbiamo aggiunto agli elementi che sono sotto questo nodo.

In questo modo possiamo semplicemente aggiornare il campo `aggiungi` e fermarci.

Ora però le nostre informazioni sono "sparse" tra i nodi dell'albero, e dobbiamo capire come recuperarle per rispondere alle query.

L'idea è che ogni volta che visitiamo un nodo, propaghiamo le informazioni di aggiunta verso i nodi figli.

Così facendo le informazioni sono sempre aggiornate e possiamo rispondere alle query come prima.

Lazy propagation

Idea

Come facciamo ad aggiornare questi nodi dell'albero?

Possiamo tenerci un nuovo campo `aggiungi` che tiene traccia di quanto abbiamo aggiunto agli elementi che sono sotto questo nodo.

In questo modo possiamo semplicemente aggiornare il campo `aggiungi` e fermarci.

Ora però le nostre informazioni sono "sparse" tra i nodi dell'albero, e dobbiamo capire come recuperarle per rispondere alle query.

L'idea è che ogni volta che visitiamo un nodo, propaghiamo le informazioni di aggiunta verso i nodi figli.

Così facendo le informazioni sono sempre aggiornate e possiamo rispondere alle query come prima.


La complessità rimane $O(\log n)$ per ogni query e update.



```
struct nodo {  
    int somma = 0;  
    int aggiungi = 0;  
};  
void propaga(int i, int tl, int tr) {  
    t[i].somma += t[i].aggiungi * (tr - tl + 1);  
    if (tl != tr) {  
        t[2 * i].aggiungi += t[i].aggiungi;  
        t[2 * i + 1].aggiungi += t[i].aggiungi;  
    }  
    t[i].aggiungi = 0;  
}
```



```
void update(int i, int tl, int tr, int l, int r, long long x) {  
    propaga(i, tl, tr);  
    if (r < tl || tr < l) return;  
    if (l <= tl && tr <= r) {  
        t[i].aggiungi += x;  
        propaga(i, tl, tr);  
        return;  
    }  
    int tm = (tl + tr) / 2;  
    update(2 * i, tl, tm, l, r, x);  
    update(2 * i + 1, tm + 1, tr, l, r, x);  
    t[i].somma = t[2 * i].somma + t[2 * i + 1].somma;  
}
```



```
int query(int i, int tl, int tr, int l, int r) {  
    propaga(i, tl, tr);  
    if (r < tl || tr < l) return 0;  
    if (l <= tl && tr <= r) return t[i].somma;  
    int tm = (tl + tr) / 2;  
    return query(2 * i, tl, tm, l, r) + query(2 * i + 1, tm + 1, tr, l, r);  
}
```

Lazy propagation

Idea

Per poter utilizzare la lazy propagation è necessario poter "unire" 2 query diverse.

In questo caso era semplice, basta sommare i due valori, ma questo non è sempre possibile / a volte richiede alcune accortezze.

Problemi

<https://cses.fi/problemset/task/1651>

<https://cses.fi/problemset/task/1734>

<https://cses.fi/problemset/task/1735>

<https://cses.fi/problemset/task/1736>

<https://training.olinfo.it/#/task/segtree/statement>

<https://training.olinfo.it/#/task/rangetree1/statement>

<https://training.olinfo.it/#/task/rangetree2/statement>

<https://training.olinfo.it/#/task/rangetree3/statement>

https://training.olinfo.it/#/task/ois_renovations/statement

https://training.olinfo.it/#/task/ois_elevator/statement