Range query e update

Segment Tree

Lorenzo Ferrari, Davide Bartoli

February 8, 2023

Table of contents

Problema motivazionale

Dynamic Range Sum Queries

Dynamic Range Sum Queries

Dato un array a di $n \leq 2 \cdot 10^5$ interi, processa $q \leq 2 \cdot 10^5$ delle seguenti query:

1. cambia in u il valore nella posizione k
2. trova la somma degli elementi nel range $[a, b]$

https://cses.fi/problemset/task/1648

Problema motivazionale

Dynamic Range Sum Queries

Dynamic Range Sum Queries

Dato un array a di $n \leq 2 \cdot 10^5$ interi, processa $q \leq 2 \cdot 10^5$ delle seguenti query:

1. cambia in u il valore nella posizione k
2. trova la somma degli elementi nel range $[a, b]$

https://cses.fi/problemset/task/1648

- possiamo risolvere il problema in maniera naive, con update in $O(1)$ e query in $O(n)$

Problema motivazionale

Dynamic Range Sum Queries

Dynamic Range Sum Queries

Dato un array a di $n \leq 2 \cdot 10^5$ interi, processa $q \leq 2 \cdot 10^5$ delle seguenti query:

1. cambia in u il valore nella posizione k
2. trova la somma degli elementi nel range $[a, b]$

https://cses.fi/problemset/task/1648

- possiamo risolvere il problema in maniera naive, con update in $O(1)$ e query in $O(n)$
- tenendoci l'array delle somme prefisse possiamo rispondere alle query in $O(1)$, ma ogni update costa $O(n)$

Problema motivazionale

Dynamic Range Sum Queries

Dynamic Range Sum Queries

Dato un array a di $n \leq 2 \cdot 10^5$ interi, processa $q \leq 2 \cdot 10^5$ delle seguenti query:

1. cambia in u il valore nella posizione k
2. trova la somma degli elementi nel range $[a, b]$

https://cses.fi/problemset/task/1648

- possiamo risolvere il problema in maniera naive, con update in $O(1)$ e query in $O(n)$
- tenendoci l'array delle somme prefisse possiamo rispondere alle query in $O(1)$, ma ogni update costa $O(n)$
- entrambe le soluzioni hanno caso pessimo $O(nq)$: al momento non sappiamo risolvere il problema efficientemente

Problema motivazionale

Sqrt Decomposition

Non possiamo ottenere complessità $O(1)$ per update **e** per query. Però possiamo ottenere complessità minore di $O(n)$ per entrambe le operazioni.

Problema motivazionale

Sqrt Decomposition

Non possiamo ottenere complessità $O(1)$ per update **e** per query. Però possiamo ottenere complessità minore di $O(n)$ per entrambe le operazioni.

Sqrt Decomposition

- dividiamo l'array in d blocchi di dimensione $\frac{n}{d}$. In ogni blocco salviamo la somma dei suoi elementi

Problema motivazionale

Sqrt Decomposition

Non possiamo ottenere complessità $O(1)$ per update **e** per query. Però possiamo ottenere complessità minore di $O(n)$ per entrambe le operazioni.

Sqrt Decomposition

- dividiamo l'array in d blocchi di dimensione $\frac{n}{d}$. In ogni blocco salviamo la somma dei suoi elementi
- per rispondere alle query, componiamo il range $[l, r]$ come somma di blocchi e di singoli elementi. Costa in totale $O(d + \frac{n}{d})$

Problema motivazionale

Sqrt Decomposition

Non possiamo ottenere complessità $O(1)$ per update **e** per query. Però possiamo ottenere complessità minore di $O(n)$ per entrambe le operazioni.

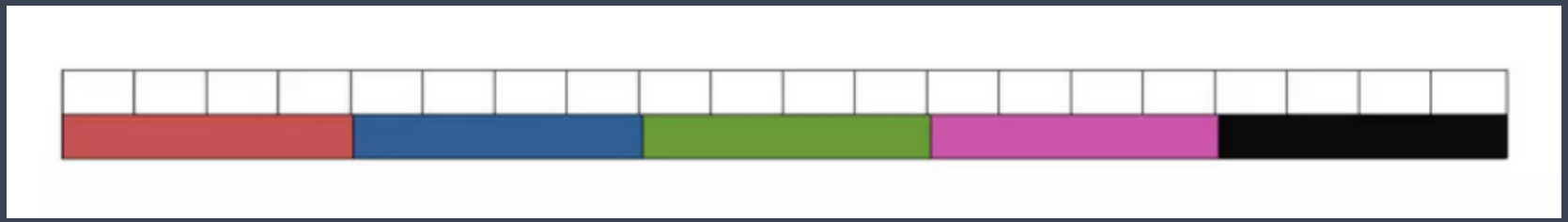
Sqrt Decomposition

- dividiamo l'array in d blocchi di dimensione $\frac{n}{d}$. In ogni blocco salviamo la somma dei suoi elementi
- per rispondere alle query, componiamo il range $[l, r]$ come somma di blocchi e di singoli elementi. Costa in totale $O(d + \frac{n}{d})$
- possiamo fare point update in $O(1)$ e range update in $O(d + \frac{n}{d})$

Problema motivazionale

Sqrt Decomposition

La scelta $d = \sqrt{n}$ dà complessità $O(\sqrt{n})$/query



Implementazione

Implementando qualunque forma di Sqrt Decomposition, **non** calcolate $d = \sqrt{n}$ al momento dell'esecuzione. Dichiarare d come una costante conosciuta a compile time permette al compilatore di ottimizzare molto meglio.

Segment Tree

Idea

Ritorniamo al problema iniziale e cerchiamo di trovare un'altra soluzione.

Segment Tree

Idea

Ritorniamo al problema iniziale e cerchiamo di trovare un'altra soluzione.
Una possibile idea è quella di salvarsi la risposta per ogni possibile intervallo.

Segment Tree

Idea

Ritorniamo al problema iniziale e cerchiamo di trovare un'altra soluzione.
Una possibile idea è quella di salvarsi la risposta per ogni possibile intervallo.
Questa soluzione peró non funziona perché i gli intervalli sono tanti ($O(N^2)$) e ogni update modifica la risposta di tanti intervalli.

Segment Tree

Idea

Ritorniamo al problema iniziale e cerchiamo di trovare un'altra soluzione.
Una possibile idea è quella di salvarsi la risposta per ogni possibile intervallo.
Questa soluzione peró non funziona perché i gli intervalli sono tanti ($O(N^2)$) e ogni update modifica la risposta di tanti intervalli.
Possiamo tenerci solo alcuni degli intevalli facendo in modo di riuscire a ricorstruire la soluzione per qualunque intervallo?

Segment Tree

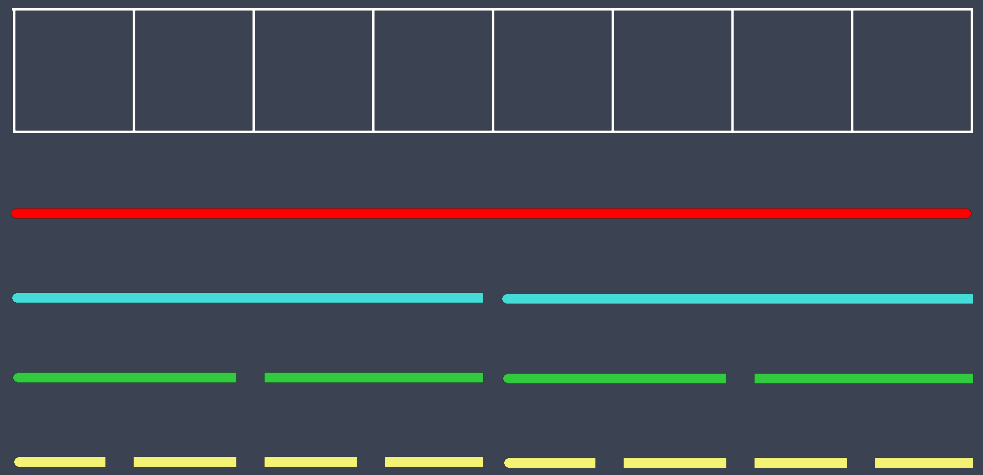
Idea

Ritorniamo al problema iniziale e cerchiamo di trovare un'altra soluzione.
Una possibile idea è quella di salvarsi la risposta per ogni possibile intervallo.
Questa soluzione peró non funziona perché i gli intervalli sono tanti ($O(N^2)$) e ogni update modifica la risposta di tanti intervalli.
Possiamo tenerci solo alcuni degli intevalli facendo in modo di riuscire a ricorstruire la soluzione per qualunque intervallo?

Segment Tree

Idea

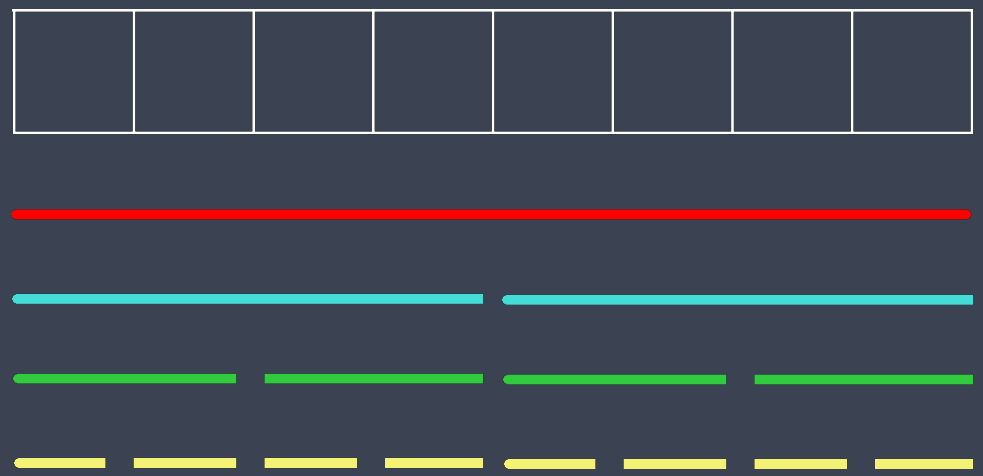
Proviamo a salvarci la risposta per alcuni intervalli la cui lunghezza è una potenza di 2.



Segment Tree

Idea

Proviamo a salvarci la risposta per alcuni intervalli la cui lunghezza è una potenza di 2.

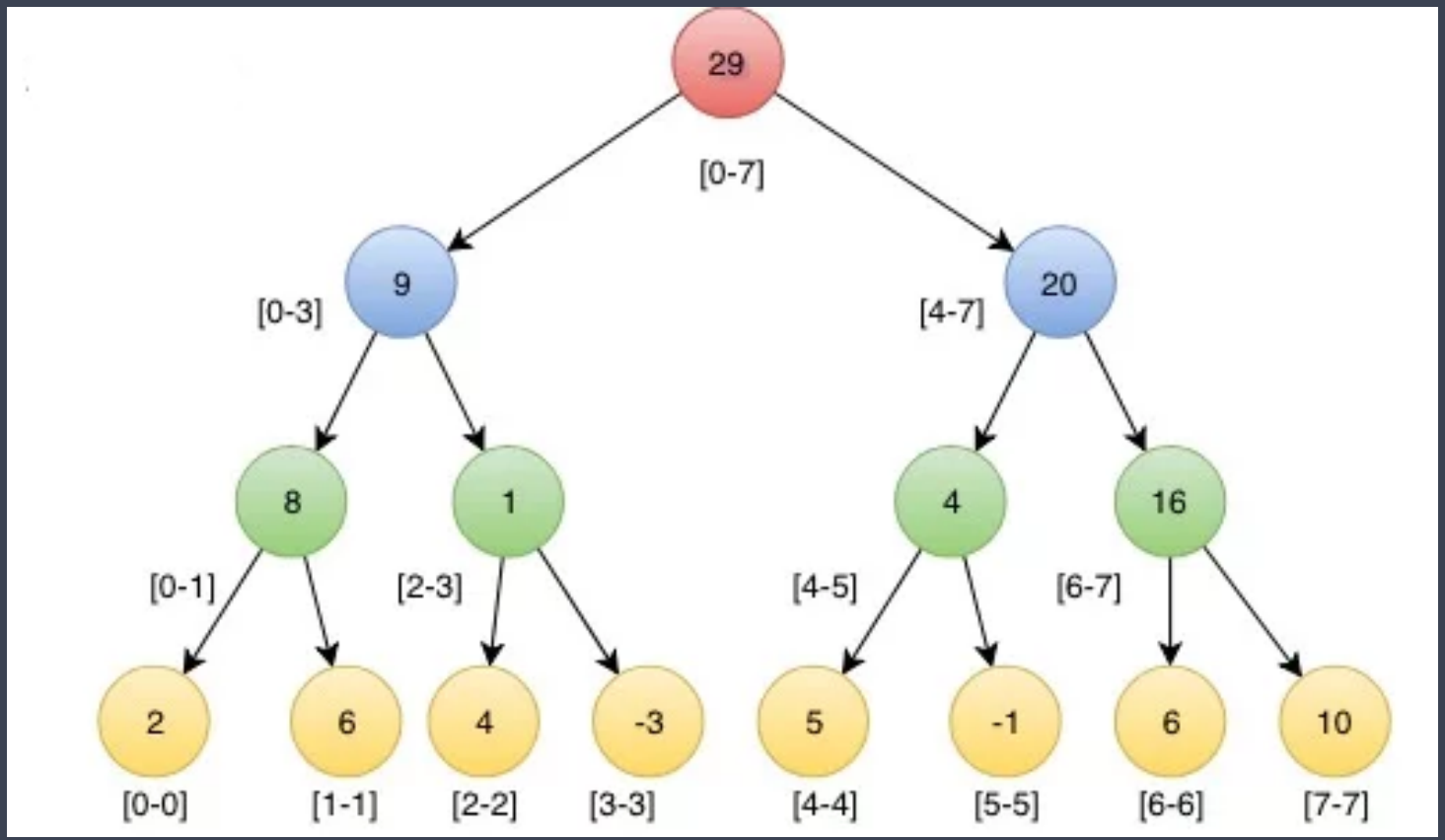


Quanti sono gli intervalli?
$N + N/2 + N/4... = 2N$

Segment Tree

Idea

Questo insieme di intervalli possiamo vederlo come un albero binario, questo ci semplificherà le cose durante l'implementazione.



Segment Tree

caratteristiche

Nominiamo i nodi a partire da 1 per livelli: la radice ha indice 1, il secondo livello contiene i nodi 2 e 3, il terzo livello i nodi 4, 5, 6 e 7 ...

Segment Tree

caratteristiche

Nominiamo i nodi a partire da 1 per livelli: la radice ha indice 1, il secondo livello contiene i nodi 2 e 3, il terzo livello i nodi 4, 5, 6 e 7 ...
Abbiamo quindi un albero con le seguenti caratteristiche:

Segment Tree

caratteristiche

Nominiamo i nodi a partire da 1 per livelli: la radice ha indice 1, il secondo livello contiene i nodi 2 e 3, il terzo livello i nodi 4, 5, 6 e 7 ...
Abbiamo quindi un albero con le seguenti caratteristiche:

- la radice ha indice 1
- il figlio sinistro di un nodo i ha indice $2i$
- il figlio destro di un nodo i ha indice $2i + 1$
- il padre di un nodo i ha indice $i/2$
- i nodi sono numerati da 1 a $2N - 1$
- l'albero ha altezza $O(\log N)$

```cpp
struct Segment {
    int n;
    vector<long long> t;
    Segment(int _n, vector<int> a) {
        for (n = 1; n < _n; n <<= 1);
        t.resize(2 * n);
        for (int i = 0; i < _n; ++i) {
            t[i + n] = a[i];
        }
        for (int i = n-1; i > 0; --i) {
            t[i] = t[2*i] + t[2*i+1];
        }
    }
};
```

Segment Tree

Update

Abbiamo definito la struttura del segment tree, ma non sappiamo ancora come utilizzarlo per risolvere il problema.

Segment Tree

Update

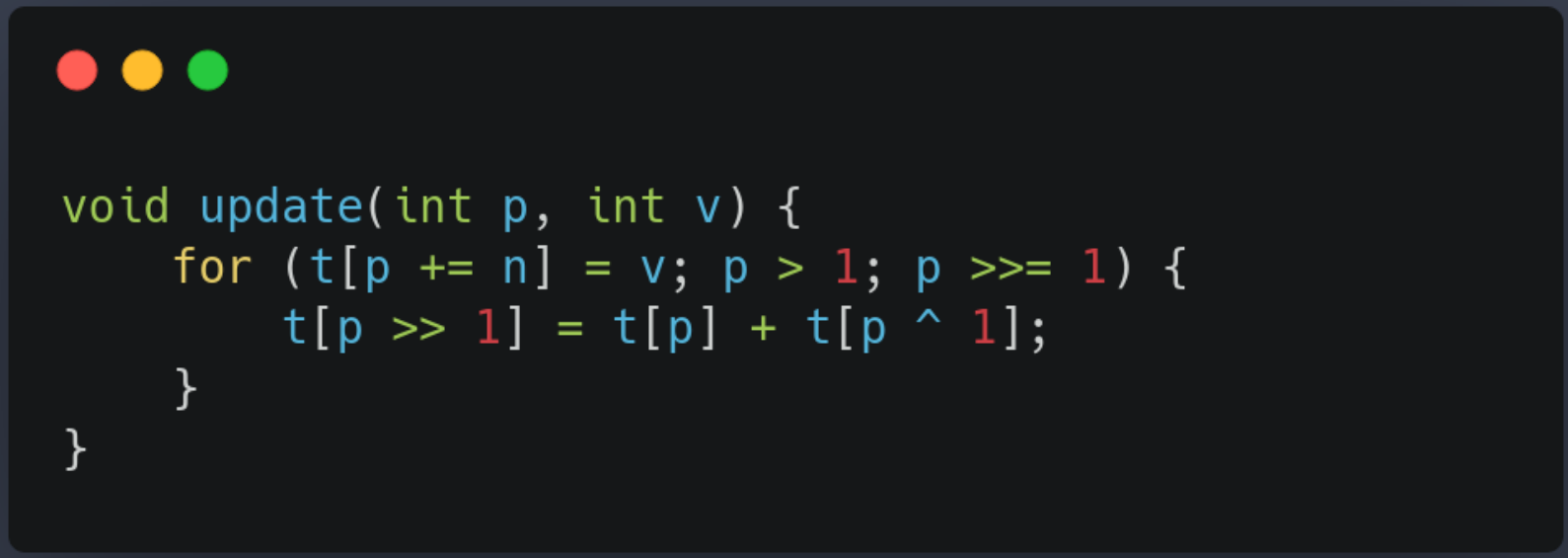
Abbiamo definito la struttura del segment tree, ma non sappiamo ancora come utilizzarlo per risolvere il problema.
Dobbiamo capire come aggiornare il segment tree e rispondere alle query.

Segment Tree

Update

Abbiamo definito la struttura del segment tree, ma non sappiamo ancora come utilizzarlo per risolvere il problema.
Dobbiamo capire come aggiornare il segment tree e rispondere alle query.
Update:
Notiamo che ogni nodo è contenuto in esattamente $\log N$ intervalli, possiamo quindi aggiornarli tutti in $O(\log N)$.

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

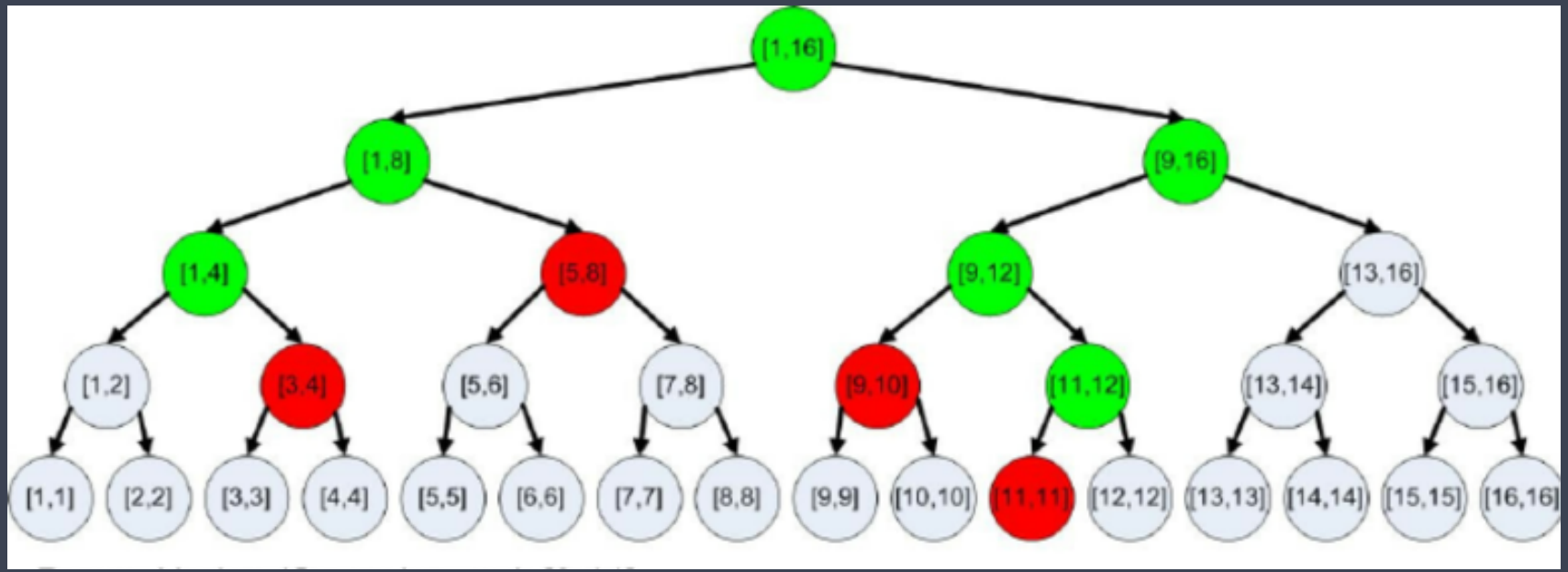


```
void update(int p, int v) {
    for (t[p += n] = v; p > 1; p >>= 1) {
        t[p >> 1] = t[p] + t[p ^ 1];
    }
}
```

Segment Tree

Query

La query è un po' più complessa. Dobbiamo trovare un insieme di intervalli da unire per ottenere la risposta desiderata.



Segment Tree

Query

Supponiamo di voler rispondere alla query $[l, r]$.
Possiamo utilizzare una dfs sull'albero. Ogni volta che raggiungiamo un nodo abbiamo 3 possibilità:

Segment Tree

Query

Supponiamo di voler rispondere alla query $[l, r]$.
Possiamo utilizzare una dfs sull'albero. Ogni volta che raggiungiamo un nodo abbiamo 3 possibilità:

- l'intervallo è completamente contenuto in $[l, r]$, quindi possiamo aggiungerlo alla risposta e fermarci
- l'intervallo è completamente fuori da $[l, r]$, quindi possiamo ignorarlo e fermarci
- l'intervallo è parzialmente contenuto in $[l, r]$, quindi ricorriamo nei figli

Segment Tree

Query

Supponiamo di voler rispondere alla query $[l, r]$.
Possiamo utilizzare una dfs sull'albero. Ogni volta che raggiungiamo un nodo abbiamo 3 possibilità:

- l'intervallo è completamente contenuto in $[l, r]$, quindi possiamo aggiungerlo alla risposta e fermarci
- l'intervallo è completamente fuori da $[l, r]$, quindi possiamo ignorarlo e fermarci
- l'intervallo è parzialmente contenuto in $[l, r]$, quindi ricorriamo nei figli

Quanti nodi visitiamo con questo processo?

Segment Tree

Query

Supponiamo di voler rispondere alla query $[l, r]$.
Possiamo utilizzare una dfs sull'albero. Ogni volta che raggiungiamo un nodo abbiamo 3 possibilità:

- l'intervallo è completamente contenuto in $[l, r]$, quindi possiamo aggiungerlo alla risposta e fermarci
- l'intervallo è completamente fuori da $[l, r]$, quindi possiamo ignorarlo e fermarci
- l'intervallo è parzialmente contenuto in $[l, r]$, quindi ricorriamo nei figli

Quanti nodi visitiamo con questo processo?
Si può dimostrare che la complessità è $O(\log N)$.
Abbiamo quindi una soluzione che risolve il problema in $O(\log N)$ per query/update.

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

Abbiamo visto un segment tree che calcola la somma in un intervallo.
Questa struttura è però molto generica e versatile, possiamo utilizzarla per risolvere molti altri problemi.

Abbiamo visto un segment tree che calcola la somma in un intervallo.
Questa struttura è però molto generica e versatile, possiamo utilizzarla per risolvere molti altri problemi.
Un esempio è utilizzarla per rispondere a query di massimo/minimo in un intervallo (come vedremo in un problema).

Problema

Parkour

Pre-Egoi Parkour

A Pisa ci sono $N \leq 10^6$ edifici, ognuno dei quali ha altezza S_i. Tommaso si trova inizialmente sul tetto dell'edificio 0 e vuole raggiungere l'edificio $N-1$ **minimizzando l'altezza massima** dei tetti su cui salta. Dalla casa i, Tommaso può raggiungere le case da $i + A_i$ a $i + B_i$ $(1 \leq A_i \leq B_i \forall i)$, dove A, B sono array in input. Aiuta Tommaso trovando la minima altezza massima per raggiungere $N-1$.

https://training.olinfo.it/#/task/pre-egoi-parkour/statement

Problema

Parkour

- senza per ora preoccuparci dell'efficienza, cerchiamo una qualunque soluzione (subesponenziale) corretta
- idee?

Problema

Parkour

- senza per ora preoccuparci dell'efficienza, cerchiamo una qualunque soluzione (subesponenziale) corretta
- idee?
- immaginiamo di essere sulla casa i e voler raggiungere $N-1$. Sia $dp[i]$ la risposta per questo sottoproblema

$$dp[i] = \max\left(S_i, \min_{j=i+A_i}^{i+B_i} dp[j]\right)$$

- la risposta al problema di partenza è $dp[0]$

Problema

Parkour

- Ci sono N stati e al momento ogni transizione costa $O(N)$. La complessità attuale è $O(N^2)$
- abbiamo appena visto una struttura dati in grado di velocizzare significativamente le transizioni!

Problema

Parkour

- Ci sono N stati e al momento ogni transizione costa $O(N)$. La complessità attuale è $O(N^2)$
- abbiamo appena visto una struttura dati in grado di velocizzare significativamente le transizioni!

```cpp
vector<int> dp(n);
st.update(n, 0);
for (int i = n-1; i >= 0; --i) {
  dp[i] = max(s[i], st.min_query(i + a[i], i + b[i]));
  st.update(i, dp[i]);
}
return dp[0];
```

Problemi

https://cses.fi/problemset/task/1648

https://cses.fi/problemset/task/1649

https://cses.fi/problemset/task/1650

https://cses.fi/problemset/task/2206

https://training.olinfo.it/#/task/pre-egoi-parkour/statement