Strutture Dati

venerdì 16 aprile 2021 16:31

Domanda: Che cos'è una DS?

Risposta: Una DS è un "contenitore" di oggetti (elementi) che permette di effettuare determinate operazioni su tali elementi.

⚠ Ogni operazione ha una complessità boundata da $O(f(size))$ per una opportuna f, dove size è la dimensione della DS.

Base di partenza

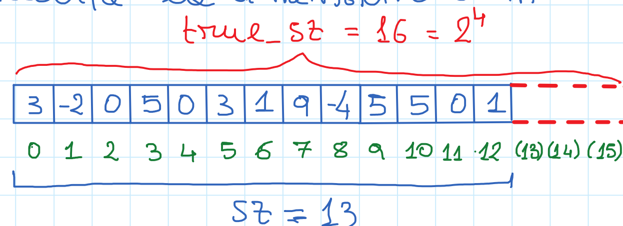
D'ora in poi, supporremo di avere una "scatola nera" (array statico) con le seguenti operazioni:

- alloc(n): alloca un array di n elementi — $O(n)$
- dealloc(): dealloca l'array e libera la memoria — $O(size)$
- at(i): accedi (in lettura e scrittura) all'elemento i-esimo — $O(1)$

VECTOR

Il vector è una DS "lineare" con le seguenti operazioni:

- push_back(v): aggiungi v in coda al vector — $O(1)$ ammortizzato
- pop_back(): rimuovi l'elemento in coda — $O(1)$
- back(): accedi all'elemento in coda — $O(1)$
- at(i): come per l'array — $O(1)$
- resize(n): resetta la dimensione a n — $O(n)$



Per implementare un vector, usiamo un array di dimensione 2^k, dove 2^k è la più piccola potenza di $2 \geq sz$.

Come gestisco resize(n)? Calcolo il nuovo valore di true_sz.

Se è $\leq$ del precedente, non faccio nulla. Altrimenti new_true_sz

- alloco un nuovo array di dimensione new_true_sz.
- copio i primi min{n, sz} elementi nel nuovo array
- dealloco il vecchio array.

QUEUE

Operazioni:

- push_back(v): inserisci elemento in fondo – O(1) ammortizzato
- pop_front(): rimuovi elemento in testa – O(1)
- front(): accedi all'elemento in testa – O(1)

?	?	6	7	7	1	2	5	2	7	3

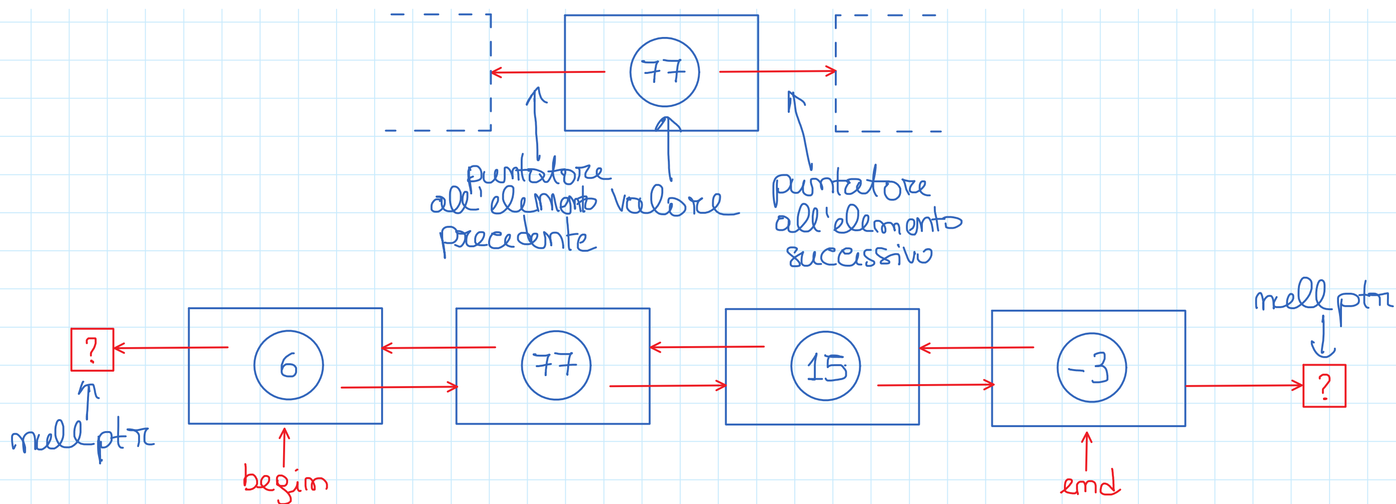
↑ end

Implemento la queue come un vector in cui tengo un "segnalino" (indice) che indica la posizione dell'elemento in fondo.
Ora, push_back(v) = vector::push_back(v), mentre pop_front() = ++ segnalino. E front() = at(segnalino).
⚠ Questa non è la STL implementation!
Esiste la deque, che è più potente della queue (ha push_front(v) e pop_back()).

LINKED LIST (DOUBLY LINKED LIST)

Operazioni:

- push_back(v), push_front(v) – O(1)
- pop_back(), pop_front() – O(1)
- at(i) – O(sz) ⚠
- reverse(): inverti la direzione della lista – O(1)
- concatenate(L): concatena L alla lista – O(1)



La struttura lista tiene soltanto i puntatori al primo e ultimo elemento (risp. begin e end).

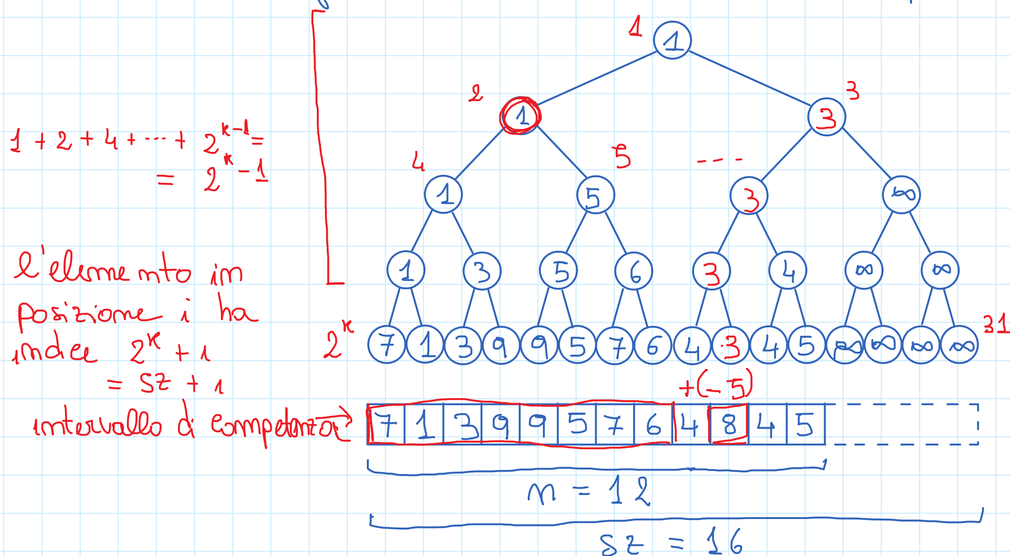
SEGMENT TREE

Problema: dato un array di interi, gestire le seguenti operazioni:

- add(i, v): aggiungi v all'elemento i-esimo
- get_min(x, y): calcola il minimo nell'intervallo [x, y).

incluso (x), escluso (y)

Posso fare meglio di $O(1)$ e $O(n)$? Sì: rispettivamente $O(\log n)$ e $O(\log n)$.

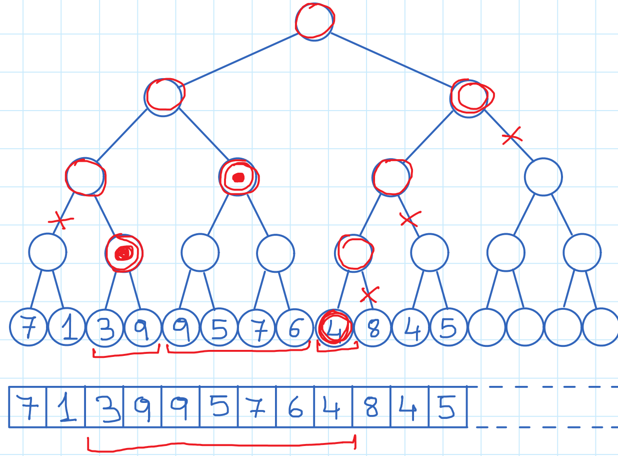


Un segment tree è un albero completo costruito sul nostro array.

In ogni nodo memorizzo il minimo del suo intervallo di competenza.

```
build(p) {
    build(2·p);
    build(2·p+1);
    st[p] = join(st[2·p], st[2·p+1]);
}
```

```
    build(2·p);
    build(2·p+1);
    st[p] = join(st[2·p], st[2·p+1]);
}
```



Domanda: quante operazioni fa get_min()? Ne fa $O(\log n)$!

Ora complichiamo le cose: se volessi gestire add(x, y, v)?

Uso la lazy-propagation.