CoderFarm - Corso base

Lezione 9

Carlo Collodel, Francesco Cerroni

16 gennaio 2023

Bentornati!

Innanzitutto, buon 2023! :)
Oggi vorremo fare il punto della situazione, rinfrescare un po' gli argomenti che abbiamo già visto e fare qualche esercizio.
Nel mezzo cercheremo di chiarire qualche dubbio e mostreremo altre funzioni utili di C++ o qualche trucchetto!

Progresso del Corso

Argomenti trattati

- Programmazione, Array, Funzioni
- Complessità asintotica
- C++: Stringhe, Containers, ...
- Ricerca binaria
- Ricorsione, Divide-et-Impera
- Greedy
- Altri algoritmi noti (Nearest Smaller El., Quickselect, ...)
- Ricerca con Backtracking, BnB

Presentazione

Quali argomenti tratteremo?

Tratteremo:

- Basi di C++.
- Metodi e funzioni più avanzate di C++, la STL.
- Algoritmi noti e tecniche varie.
- Tecniche Greedy.
- Complete search e Backtracking.
- Programmazione dinamica.
- Algoritmi su grafi.
- Strutture dati avanzate.

Recap fondamentale

Complessità asintotica

La complessità asintotica è un modo molto utile per descrivere l'efficienza di un algoritmo. In particolare se abbiamo un algoritmo con complessità $\mathcal{O}(N^2)$ significa che il numero di operazioni è proporzionale a N^2 ("nell'ordine di N^2 operazioni").
Le regole principali da ricordare sono:

- Le costanti sommate si possono omettere
- Sommando due complessità si mantiene la funzione che "cresce di più" per N grandi
- Stiamo attenti alle costanti moltiplicative significative

Recap fondamentale

Complessità asintotica

Esempi

- Ricerca binaria in un array: $\mathcal{O}(\log n)$
- Ricerca binaria e algoritmo lineare: $\mathcal{O}(n \log n)$
- Enumerare tutte le permutazioni di un array: $\mathcal{O}(n!)$
- Quickselect: $\mathcal{O}(2n) \approx \mathcal{O}(n)$
- `std::sort` + Greedy Lineare: $\mathcal{O}(n \log n)$

Programmazione

Weird Algorithm (cses.fi)

Il problema riguarda la *Congettura di Collatz*:
Viene dato un numero N, se il numero è pari dividere per 2, se è dispari moltiplicare per 3 e aggiungere 1, ripetere finchè non si ha raggiunto 1.

Programmazione

Increasing Array (cses.fi)

Viene dato un array di N interi, si vuole modificare questo array in modo tale che ogni elemento sia grande almeno quanto il precedente.
Con una mossa si può aumentare un numero di 1, qual è il numero minimo di mosse necessarie?

Errori frequenti

Dai *Vincoli* del problema notiamo che:

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq x_i \leq 10^9$ (elementi dell'array)

Consideriamo il caso con $x = [10^9, 1, 1, 1, 1, ..., 1]$, serve un `long long` per il risultato! ($\texttt{res} \approx (N-1) \cdot 10^9$)

Ordinamento

Palindrome Reorder

Data una stringa S, riordinarla in modo tale che diventi **palindroma** (si legge uguale da sinistra e da destra).

Anagrammi, palindromi, ...

Spesso, ci può capitare di trovare problemi che riguardano argomenti sullo spostare caratteri o analizzarne la frequenza: l'ordinamento ci può spesso aiutare in queste situazioni! Ad esempio: come verifichiamo se una stringa A è un anagramma della stringa B? Possiamo ordinare ciascuna stringa carattere per carattere e confrontare se le due stringhe sono uguali (oppure conto la frequenza delle lettere).

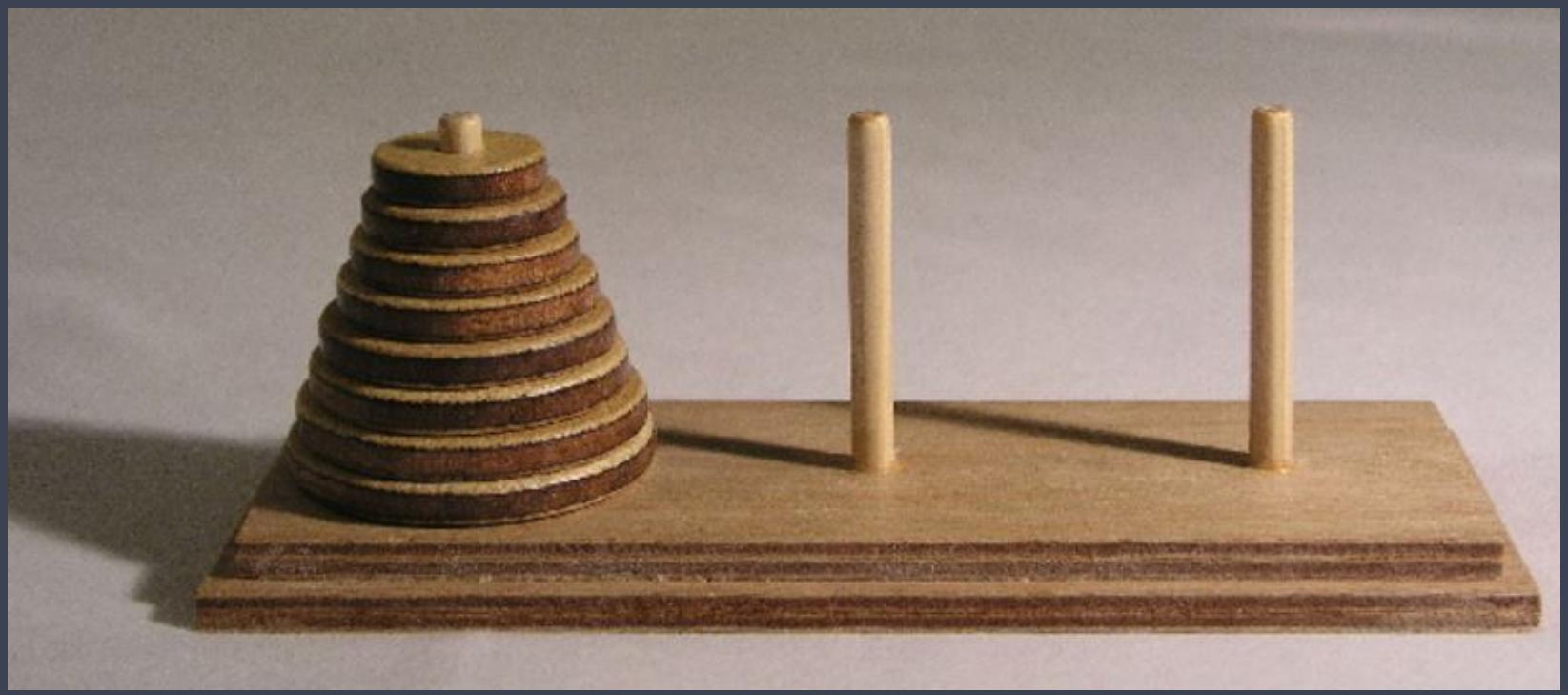
Ordinamento

Palindrome Reorder

Per risolvere questo problema, basta controllare che ogni carattere appaia un numero pari di volte e bisogna controllare un'altra cosa, provate a pensarci e scrivete le vostre idee in chat!
Appena sappiamo che una stringa può essere riordinata in una palindroma, possiamo generare facilmente una risposta valida.

Ricorsione

Tower Of Hanoi



Ricorsione

Tower Of Hanoi

Vengono date tre pile dove si possono posizionare oggetti e N dischi di dimensioni diverse tra loro.
Inizialmente tutti i dischi sono sulla prima pila ordinati dal più piccolo in alto al più grande in basso.
È possibile spostare un disco che si trova in cima a una delle tre pile in cima a un'altra pila, a patto che non si stia piazzando un disco più grande sopra a un disco più piccolo.
Trovare quante e quali sono le mosse minime per spostare tutti i dischi dalla prima pila all'ultima usando quella centrale!

Ricorsione

Tower Of Hanoi

Proviamo a trovare una ricorrenza che ci aiuti a risolvere il problema.

Osservazione importante

Supponiamo di voler spostare i più piccoli $K \leq N$ dischi sull'ultima pila, allora non è importante sapere su che pila si trovano i restanti $N - K$ dischi, perché questi saranno tutti più grandi dei K dischi che vogliamo spostare!

Il requisito che ci serve per spostare il K-esimo disco sull'ultima pila è che non ci siano dischi *sopra* di lui, quindi potrei pensare di **spostare i primi $K - 1$ dischi dalla prima pila alla pila in mezzo**, poi potrò finalmente spostare il K-esimo disco.
Ma così il problema si riduce a **spostare i primi $K - 1$ dischi dalla pila in mezzo all'ultima pila**!

In altre parole, se voglio spostare N dischi dalla prima pila all'ultima pila, posso:

- Spostare i primi $N-1$ dischi che si trovano sulla prima pila in quella centrale (in questo modo avrò solo 1 disco sulla prima pila, quello più grande)
- Spostare l'unico disco che si trova sulla prima pila nell'ultima pila
- Spostare gli $N-1$ dischi che si trovano nella pila centrale sopra il disco più grande (quindi sull'ultima pila)

Ricorsione

Tower Of Hanoi

Se volessi scriverlo come ricorrenza matematica (Numero di mosse per spostare N dischi da A a C, usando B come appoggio), potrei scrivere:

$$T(N, A, B, C) = \begin{cases} 0 & \text{se } N = 0 \\ T(N-1, A, C, B) + 1 + T(N-1, B, A, C) & \text{se } N > 0 \end{cases}$$

Ricorsione

Soluzione Tris

Problema: tris.
(`https://training.olinfo.it/#/task/ois_tris/statement`)

Soluzione

Il numero di celle è abbastanza piccolo da poter applicare un algoritmo brute force: generiamo tutti i possibili stati del gioco. L'importante per stare nei tempi è notare che ogni volta che è presente un tris la partita si interrompe e non serve calcolare oltre.

Ricorsione

Cabala

Problema: il numero della cabala.
(`https://training.olinfo.it/#/task/ois_cabala/statement`)

Soluzione

Possiamo provare tutti i possibili valori conformi alle richiste. La prima cifra del numero può essere 3, 6, oppure 9, e una volta fissata procediamo a aggiungere cifre. Ogni voltà però abbiamo 2 possibilità di scelta: non possiamo ripetere la cifra precedente. Il numero di valori da generare è quindi $3 \cdot 2^{N-1}$, che nel caso pessimo è $3 \cdot 2^{17} = 393216$ (molto poco).

Ricorsione

Chessboard and Queens

Mostra codice

Ricorsione

Grid Paths

Mostra codice

Sorting and Searching

Restaurant Customers

Vengono dati i tempi di *entrata* e di *uscita* di N clienti in un ristorante, trova il numero massimo di clienti presenti contemporaneamente nel ristorante in un qualsiasi momento.

Osservazione

Se un cliente esce nello stesso momento in cui entra un altro cliente, conto entrambi i clienti come presenti?
Fortunatamente nella descrizione del problema c'è scritto che "Possiamo assumere che tutti i tempi di entrata e uscita sono distinti".

Sorting and Searching

Restaurant Customers

Una prima idea potrebbe essere la seguente:

- Creo un array grande che può contenere il massimo tempo di entrata/uscita possibile (con valori a 0)
- Per ogni cliente, aggiungo 1 nell'array all'indice rispettivo al tempo di entrata, tolgo 1 nella posizione del tempo di uscita
- Quando ho finito, itero per ogni elemento dell'array e mantengo una somma degli elementi considerati fin'ora: ogni volta che aggiungo un elemento alla somma controllo se la mia somma corrente è massima e tengo sempre la più grande.

La complessità è lineare nei tempi di entrata e uscita massimi.

Sorting and Searching

Restaurant Customers

Errori frequenti

L'algoritmo che abbiamo visto è lineare, quindi può sembrare che sia ottimale per il nostro problema, ma se osserviamo i *Constraints* nel testo del problema, possiamo notare che $1 \leq a < b \leq 10^9$ (dati a, b rispettivamente tempo di entrata e uscita).
Quindi il nostro algoritmo è lineare, ma a e b sono troppo grandi per ammettere un algoritmo lineare sulla loro dimensione![1]

Pensiamo a un algoritmo alternativo...

[1] Se usassi una `std::map` invece, potrei usare questo algoritmo!

Sorting and Searching

Restaurant Customers

- Creo un `vector<pair<int, int>>` in cui salvo i tempi di entrata e uscita di ogni cliente, uso il primo `int` per il valore del tempo e nel secondo `int` inserisco 1 o -1 per distinguere un'entrata o un'uscita.
- Ordino il vettore.
- Come prima mantengo una somma iterando per ogni elemento del vettore, sommo 1 in caso di entrata e -1 in caso di uscita (mi basta sommare il secondo campo del `pair`) e poi tengo la somma massima come risposta.

La complessità è $\mathcal{O}(n \log n)$, che è **peggiore** di una complessità lineare, però dai *Constraints* osserviamo che $N \leq 2 \cdot 10^5$: questo algoritmo sta nei tempi di esecuzione!

Greedy

Towers

Vengono dati N cubi di dimensioni diverse in un certo ordine. Qual è il numero minimo di *torri* che si possono creare usando questi cubi? Una *torre* è composta da diversi cubi posizionati uno sopra l'altro, ogni cubo deve essere più piccolo del cubo sotto di lui.

Greedy

Towers

Quando conviene creare una torre nuova al posto di piazzare un cubo su una torre già esistente?

Torri nuove?

Supponiamo di dover piazzare un cubo con dimensione X e di avere una torre già costruita il cui cubo in cima ha dimensione Y (con $X < Y$, in altre parole posso piazzare X sopra Y).

- Se creo una nuova torre, avrò due torri, con gli elementi in cima rispettivamente X e Y.
- Se non creo una nuova torre, avrò una sola torre con l'elemento in cima X.

Greedy

Towers

Se ho più torri dove posso piazzare un cubo, in quale di queste mi conviene piazzarlo?

Più torri?

Supponiamo di dover piazzare un cubo con dimensione X e di avere due torri con cubi in cima di dimensioni Y, Z (con $X < Y < Z$).

- Se piazzo X sopra Y, ottengo due torri con X, Z in cima.
- Se piazzo X sopra Z, ottengo due torri con X, Y in cima.

Binary Search The Answer

Array Division

Viene dato un array di N interi positivi.
Dividere l'array in K sottoarray (contigui) in modo che il massimo tra le somme degli elementi di ogni sottoarray è minimo.

Un problema più semplice

Questo problema sembra molto complicato all'inizio, pensiamo a un problema un po' più semplice!
Viene dato un array di N interi positivi.
Dividere l'array in K sottoarray (contigui) in modo che il massimo tra le somme degli elementi di ogni sottoarray non superi M.

Binary Search The Answer

Array Division

Nel caso del problema semplificato, ci basta iterare sull'array e creare dei sottoarray aggiungendo elementi finchè non si sta per superare somma M.
Poi conto il numero di sottoarray creati in questo modo:

- Se questi sono più di K allora sicuramente non esiste soluzione a questo problema
- In caso opposto, posso sempre spezzare i sottoarray creati fino ad ottenere una suddivisione in esattamente K sottoarray.

Binary Search The Answer

Array Division

Osservazione

Se posso costruire un array valido per M, quell'array è valido anche per qualsiasi valore $\geq M$.

Allora posso fare **ricerca binaria** sul valore di M, trovando il *più piccolo M che ha soluzione*!
Questo tipo di algoritmi si chiama **Binary Search The Answer**: letteralmente si fa ricerca binaria sulla risposta del problema e si controlla se è valida.

Esercizi per casa!

- Repetitions (difficoltà Facile)
  `https://cses.fi/problemset/task/1625`
- Two Sets (difficoltà Medio)
  `https://cses.fi/problemset/task/1092`
- Gray Code (difficoltà Difficile)
  `https://cses.fi/problemset/task/2205`
- Distinct Numbers (difficoltà Facile)
  `https://cses.fi/problemset/task/1621`
- Sum of Two Values (difficoltà Media)
  `https://cses.fi/problemset/task/1640`
- Maximum Subarray Sum (difficoltà Media)
  `https://cses.fi/problemset/task/1643`
- Subarray Distinct Values (difficoltà Difficile)
  `https://cses.fi/problemset/task/2428`

Fine

- **E-Mail**: `base@coderfarm.it`