CoderFarm - Corso avanzato

Lezione 0

Lorenzo Ferrari, Davide Bartoli

16 settembre 2023

Presentazione

A chi è rivolto il corso?

Il corso avanzato è per chi sa già programmare e vuole avvicinarsi al mondo della Programmazione Competitiva (CP).
Il nostro obiettivo è prepararvi per le Olimpiadi di Informatica (OIS/OII) e in generale per le gare di CP.

Vi ricordiamo che:

- In questo corso daremo per scontata una conoscenza di base della programmazione C++
- Le lezioni si svolgeranno con cadenza settimanale, il Mercoledì dalle 16.00

Presentazione

Quali argomenti tratteremo?

Tratterემო:

- Complessità computazionale (a fini pratici)
- Ricorsione, Complete search e Backtracking.
- Programmazione dinamica
- Algoritmi sui Grafi
- Algoritmi greedy
- Strutture dati avanzate
- Algoritmi probabilistici
- Tecniche e “trucchi” anche inusuali

Il livello delle lezioni potrà essere aggiustato in base al vostro feedback.

Presentazione

Chi siamo?

- **Nome:** Lorenzo Ferrari
- **Classe:** 2004
- **Provenienza:** Trento
- **Risultati alle OII:** due ori e un argento, primo posto OII 2022
- **Altri risultati olimpici:** menzione d'onore IOI, prima squadra italiana alle EOES

Presentazione

Chi siamo?

- **Nome:** Davide Bartoli
- **Classe:** 2003
- **Provenienza:** Imola
- **Risultati alle OII:** tre ori, primo posto OII 2020 e 2021
- **Altri risultati olimpici:** argento IOI, due ori alle olimpiadi di matematica

Introduzione

Gare di Informatica

Come funziona una gara di Informatica?

- Dati i problemi, si scrivono dei programmi[1] che
  - prendono dei dati in *input*;
  - elaborano i dati per risolvere il problema;
  - ritornano i risultati in *output*.
- I programmi devono produrre la risposta corretta entro limiti di tempo e di memoria.
- Spesso i problemi hanno dei *subtask* che forniscono *punti parziali* e guidano verso la soluzione del problema.

[1] quasi sempre in C++

Introduzione

Gare di Informatica

Una soluzione deve essere:

Introduzione

Gare di Informatica

Una soluzione deve essere:

- **Corretta**: deve risolvere il problema

Introduzione

Gare di Informatica

Una soluzione deve essere:

- **Corretta**: deve risolvere il problema
- **Efficiente**: deve essere abbastanza veloce da trovare la soluzione entro il tempo limite

Introduzione

Gare di Informatica

Una soluzione deve essere:

- **Corretta**: deve risolvere il problema
- **Efficiente**: deve essere abbastanza veloce da trovare la soluzione entro il tempo limite

Per controllare se una soluzione è efficiente possiamo stimare il numero di operazioni che il programma esegue calcolando la **complessità computazionale** del programma, ovvero una stima del numero di operazioni eseguite dal codice in funzione della dimensione dell'input.

Introduzione

Gare di Informatica

Una soluzione deve essere:

- **Corretta**: deve risolvere il problema
- **Efficiente**: deve essere abbastanza veloce da trovare la soluzione entro il tempo limite

Per controllare se una soluzione è efficiente possiamo stimare il numero di operazioni che il programma esegue calcolando la **complessità computazionale** del programma, ovvero una stima del numero di operazioni eseguite dal codice in funzione della dimensione dell'input.

Calcolare la complessità computazionale di un algoritmo permette di valutare se la soluzione è abbastanza veloce per risolvere il problema prima ancora di scrivere il codice, facendo risparmiare tempo e fatica.

Complessità Computazionale

introduzione

La complessità computazionale si esprime attraverso la notazione $O(f(N))$[2], dove $f(N)$ è una funzione che stima il numero di operazioni eseguite dal programma in funzione della dimensione dell'input N.

[2] letto "O grande di $f(N)$"

Complessità Computazionale

introduzione

La complessità computazionale si esprime attraverso la notazione $O(f(N))$[2], dove $f(N)$ è una funzione che stima il numero di operazioni eseguite dal programma in funzione della dimensione dell'input N.

Esempi:

- $O(1)$, l'algoritmo esegue sempre lo stesso numero di operazioni indipendentemente dall'input

[2] letto "O grande di $f(N)$"

Complessità Computazionale

introduzione

La complessità computazionale si esprime attraverso la notazione $O(f(N))$[2], dove $f(N)$ è una funzione che stima il numero di operazioni eseguite dal programma in funzione della dimensione dell'input N.

Esempi:

- $O(1)$, l'algoritmo esegue sempre lo stesso numero di operazioni indipendentemente dall'input
- $O(\log_2 N)$, esempio più tardi

[2] letto "O grande di $f(N)$"

Complessità Computazionale

introduzione

La complessità computazionale si esprime attraverso la notazione $O(f(N))$[2], dove $f(N)$ è una funzione che stima il numero di operazioni eseguite dal programma in funzione della dimensione dell'input N.

Esempi:

- $O(1)$, l'algoritmo esegue sempre lo stesso numero di operazioni indipendentemente dall'input
- $O(\log_2 N)$, esempio più tardi
- $O(N)$, per esempio iterare sugli elementi di un array di dimensione N

[2] letto "O grande di $f(N)$"

Complessità Computazionale

La complessità computazionale si esprime attraverso la notazione $O(f(N))$[2], dove $f(N)$ è una funzione che stima il numero di operazioni eseguite dal programma in funzione della dimensione dell'input N.

Esempi:

- $O(1)$, l'algoritmo esegue sempre lo stesso numero di operazioni indipendentemente dall'input
- $O(\log_2 N)$, esempio più tardi
- $O(N)$, per esempio iterare sugli elementi di un array di dimensione N
- $O(N^2)$, per esempio iterare sulle coppie di elementi

[2] letto "O grande di $f(N)$"

Complessità Computazionale

introduzione

La complessità computazionale si esprime attraverso la notazione $O(f(N))$[2], dove $f(N)$ è una funzione che stima il numero di operazioni eseguite dal programma in funzione della dimensione dell'input N.

Esempi:

- $O(1)$, l'algoritmo esegue sempre lo stesso numero di operazioni indipendentemente dall'input
- $O(\log_2 N)$, esempio più tardi
- $O(N)$, per esempio iterare sugli elementi di un array di dimensione N
- $O(N^2)$, per esempio iterare sulle coppie di elementi
- $O(2^N)$, per esempio iterare su tutti i sottoinsiemi

[2] letto "O grande di $f(N)$"

Complessità Computazionale

Caratteristiche

- le operazioni elementari ($+$, *, -, $/$, $\ldots$) sono considerate costanti: $O(1)$

Complessità Computazionale

Caratteristiche

- le operazioni elementari ($+$, *, -, $/$, $\ldots$) sono considerate costanti: $O(1)$
- dichiarazioni, assegnamenti, confronti di variabili semplici (`int`, `float`, $\ldots$) sono considerati costanti: $O(1)$

- le operazioni elementari ($+$, *, -, $/$, ...) sono considerate costanti: $O(1)$
- dichiarazioni, assegnamenti, confronti di variabili semplici (`int`, `float`, ...) sono considerati costanti: $O(1)$
- input e output di variabili semplici sono considerati costanti: $O(1)$

Complessità Computazionale

Caratteristiche

- le operazioni elementari (+, *, -, /, ...) sono considerate costanti: $O(1)$
- dichiarazioni, assegnamenti, confronti di variabili semplici (`int`, `float`, ...) sono considerati costanti: $O(1)$
- input e output di variabili semplici sono considerati costanti: $O(1)$
- ci interessano solo i termini più grandi: $O(N^2 + N + 1) = O(N^2)$

Complessità Computazionale

Caratteristiche

- le operazioni elementari ($+$, *, -, $/$, $\ldots$) sono considerate costanti: $O(1)$
- dichiarazioni, assegnamenti, confronti di variabili semplici (`int`, `float`, $\ldots$) sono considerati costanti: $O(1)$
- input e output di variabili semplici sono considerati costanti: $O(1)$
- ci interessano solo i termini più grandi: $O(N^2 + N + 1) = O(N^2)$
- non ci interessano i fattori costanti: $O(10 \cdot N^2) = O(N^2)$

Esempio di calcolo della complessità

```cpp
cin >> N;                   // O(1)
int somma = 0;              // O(1)

// eseguito O(N) volte
for (int i = 0; i < N; i++){
  int a;                    // O(1)
  cin >> a;                 // O(1)
  somma += a;               // O(1)
}

cout<<somma<<endl;          // O(1)

// totale O(1+1+ N*(1+1+1) +1)
// = O(2+3*N+1) = O(N)
```

Esempio di calcolo della complessità

```cpp
cin >> N;                   // O(1)
int somma = 0;              // O(1)

// eseguito O(N) volte
for (int i = 0; i < N; i++){

  // eseguito O(N-i) volte
  for (int j = i; i < N; i++){
    somma += i*j;           // O(1)
  }
}

cout << somma << endl;  // O(1)

// totale O(1+1+ N*(N+1)/2 +1)
// = O(3+N*(N+1)/2) = O(N^2)
```

Introduzione

Problema di esempio

Lotteria di quadri

Data una sequenza di $N \leq 200\,000$ interi positivi e un intero M, trovare il massimo $B \leq N$ tale che la somma di ogni sottosegmento lungo B sia al più M.

Introduzione

Problema di esempio

Lotteria di quadri

Data una sequenza di $N \leq 200\,000$ interi positivi e un intero M, trovare il massimo $B \leq N$ tale che la somma di ogni sottosegmento lungo B sia al più M.

Come facciamo a controllare se un B va bene?

Introduzione

Problema di esempio

Lotteria di quadri

Data una sequenza di $N \leq 200\,000$ interi positivi e un intero M, trovare il massimo $B \leq N$ tale che la somma di ogni sottosegmento lungo B sia al più M.

Come facciamo a controllare se un B va bene?

- per ogni $i = 0, 1, \ldots, N - B$ possiamo controllare se la somma di $A[i], A[i+1], \ldots, A[i + B - 1]$ è al più M in tempo $O(B)$. Dato che dobbiamo controllare ogni valore di i, il tempo totale è $O(NB)$. Questa soluzione è troppo lenta, cerchiamo di renderla più veloce.

Controllo in $O(NB)$

Implementazione

```cpp
bool works(int b) {
  for (int i = 0; i < n - b; i++) {
    long long sum = 0;
    for (int j = i; j < i + b; j++) {
      sum += v[j];
    }
    if (sum > m) return false;
  }
  return true;
}
```

Introduzione

Problema di esempio

Possiamo fare meglio?

Introduzione

Problema di esempio

Possiamo fare meglio?

- possiamo notare che stiamo calcolando più volte le stesse somme. In particolare conoscendo la somma di $A[i], A[i+1], \ldots, A[i+B-1]$ possiamo facilmente calcolare la somma di $A[i+1], A[i+2], \ldots, A[i+B]$ senza dover ricalcolare tutto da capo.

Possiamo fare meglio?

- possiamo notare che stiamo calcolando più volte le stesse somme. In particolare conoscendo la somma di $A[i], A[i+1], \ldots, A[i+B-1]$ possiamo facilmente calcolare la somma di $A[i+1], A[i+2], \ldots, A[i+B]$ senza dover ricalcolare tutto da capo.
- chiamiamo K la somma di $A[i], A[i+1], \ldots, A[i+B-1]$. Allora possiamo calcolare la somma di $A[i+1], A[i+2], \ldots, A[i+B]$ in tempo $O(1)$ come $K - A[i] + A[i+B]$.

Introduzione

Problema di esempio

Possiamo fare meglio?

- possiamo notare che stiamo calcolando più volte le stesse somme. In particolare conoscendo la somma di $A[i], A[i+1], \ldots, A[i+B-1]$ possiamo facilmente calcolare la somma di $A[i+1], A[i+2], \ldots, A[i+B]$ senza dover ricalcolare tutto da capo.
- chiamiamo K la somma di $A[i], A[i+1], \ldots, A[i+B-1]$. Allora possiamo calcolare la somma di $A[i+1], A[i+2], \ldots, A[i+B]$ in tempo $O(1)$ come $K - A[i] + A[i+B]$.
- in questo modo possiamo controllare se un B va bene in tempo $O(B + (N-B)) = O(N)$.

Controllo in $O(N)$

Implementazione

```
bool works(int b) {
  long long sum = 0;
  for (int i = 0; i < b; ++i)
    sum += v[i];

  long long max_sum = sum;
  for (int i = b; i < n; ++i) {
    sum += v[i] - v[i - b];
    max_sum = max(max_sum, sum);
  }

  return max_sum <= m;
}
```

Introduzione

Problema di esempio

- ora sappiamo controllare se un B va bene in tempo $O(N)$. Come possiamo trovare il massimo B che va bene?

Introduzione

Problema di esempio

- ora sappiamo controllare se un B va bene in tempo $O(N)$. Come possiamo trovare il massimo B che va bene?
- possiamo controllare $B = j$ per $j = 0, 1, \ldots, N$ e prendere il più grande valore valido. Questo ha complessità $O(N^2)$

Introduzione

Problema di esempio

- ora sappiamo controllare se un B va bene in tempo $O(N)$. Come possiamo trovare il massimo B che va bene?
- possiamo controllare $B = j$ per $j = 0, 1, \ldots, N$ e prendere il più grande valore valido. Questo ha complessità $O(N^2)$
- ahimè $(2 \cdot 10^5)^2$ operazioni sono decisamente troppe per 2 secondi

Introduzione

Problema di esempio

- ora sappiamo controllare se un B va bene in tempo $O(N)$. Come possiamo trovare il massimo B che va bene?
- possiamo controllare $B = j$ per $j = 0, 1, \dots, N$ e prendere il più grande valore valido. Questo ha complessità $O(N^2)$
- ahimè $(2 \cdot 10^5)^2$ operazioni sono decisamente troppe per 2 secondi
- si può fare meglio di così?

Lotteria di quadri

Data una sequenza di $N \leq 200\,000$ interi positivi e un intero M, trovare il massimo $B \leq N$ tale che la somma di ogni sottosegmento lungo B sia al più M.

Osservazioni:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Introduzione

Problema di esempio

Lotteria di quadri

Data una sequenza di $N \leq 200\,000$ *interi positivi e un intero* M*, trovare il massimo* $B \leq N$ *tale che la somma di ogni sottosegmento lungo* B *sia al più* M*.*

Osservazioni:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Con queste premesse esiste un algoritmo che ci permette di trovare il più grande B_x valido con pochi confronti!

Ricerca binaria

Algoritmo

Definizione del problema

Dato un array potenzialmente molto grande della forma $[\ldots, 0, 0, 0, 1, 1, 1, \ldots]$, trova l'ultimo 0.

Definizione del problema

Dato un array potenzialmente molto grande della forma $[\ldots, 0, 0, 0, 1, 1, 1, \ldots]$, trova l'ultimo 0.

Una possibile soluzione è controllare tutti gli elementi dell'array in ordine e fermarci quando troviamo un 1. Qusta soluzione è corretta ma non è efficiente, infatti ha complessità $O(N)$. Possiamo fare meglio, sfruttando il fatto che tutti gli 0 sono prima di tutti gli 1?

https://forum.olinfo.it/t/7224

Ricerca binaria

Algoritmo

Idea

Per cercare una determinata pagina in un libro nessuno scorre pagina per pagina dall'inizio. È molto più pratico aprire circa a metà, controllare in che metà si trova la pagina che cerchiamo, e così finché non abbiamo finito.

Ricerca binaria

Algoritmo

Idea

Per cercare una determinata pagina in un libro nessuno scorre pagina per pagina dall'inizio. È molto più pratico aprire circa a metà, controllare in che metà si trova la pagina che cerchiamo, e così finché non abbiamo finito.

Algoritmo:

- iniziamo con un intervallo $[l, r)$

Ricerca binaria

Algoritmo

Idea

Per cercare una determinata pagina in un libro nessuno scorre pagina per pagina dall'inizio. È molto più pratico aprire circa a metà, controllare in che metà si trova la pagina che cerchiamo, e così finché non abbiamo finito.

Algoritmo:

- iniziamo con un intervallo $[l, r)$
- controlliamo $mid = (l + r)/2$
- se mid è 0, allora la risposta si trova in $[mid, r)$
- altrimenti la risposta si trova in $[l, mid)$

Ricerca binaria

Algoritmo

Idea

Per cercare una determinata pagina in un libro nessuno scorre pagina per pagina dall'inizio. È molto più pratico aprire circa a metà, controllare in che metà si trova la pagina che cerchiamo, e così finché non abbiamo finito.

Algoritmo:

- iniziamo con un intervallo $[l, r)$
- controlliamo $mid = (l + r)/2$
- se mid è 0, allora la risposta si trova in $[mid, r)$
- altrimenti la risposta si trova in $[l, mid)$

Ripetiamo questo processo finchè l'intervallo $[l, r)$ non ha dimensione 1 (ovvero $l = r - 1$). L'unico elemento rimasto è la risposta che stavamo cercando.

Ricerca binaria

Complessità di tempo

- Lo spazio di ricerca è inizialmente N, poi viene dimezzato ad ogni iterazione finché non rimane un solo elemento.

Ricerca binaria

Complessità di tempo

- Lo spazio di ricerca è inizialmente N, poi viene dimezzato ad ogni iterazione finché non rimane un solo elemento.
- Quindi inizialmente abbiamo N possibili candidati, poi $N/2$, poi $N/4$, e così via.

Ricerca binaria

Complessità di tempo

- Lo spazio di ricerca è inizialmente N, poi viene dimezzato ad ogni iterazione finché non rimane un solo elemento.
- Quindi inizialmente abbiamo N possibili candidati, poi $N/2$, poi $N/4$, e così via.
- In totale sono sufficienti $\lceil \log_2 N \rceil$ iterazioni e controlli.

Logaritmo

Il logaritmo in base 2 di N è il numero di volte che bisogna moltiplicare 2 per ottenere N.

$$\log_2 N = x \iff 2^x = N$$

Questo valore cresce molto lentamente, per esempio $\log_2 10^6 \approx 20$.

Ricerca binaria

Complessità di tempo

- Lo spazio di ricerca è inizialmente N, poi viene dimezzato ad ogni iterazione finché non rimane un solo elemento.
- Quindi inizialmente abbiamo N possibili candidati, poi $N/2$, poi $N/4$, e così via.
- In totale sono sufficienti $\lceil \log_2 N \rceil$ iterazioni e controlli.

Logaritmo

Il logaritmo in base 2 di N è il numero di volte che bisogna moltiplicare 2 per ottenere N.

$$\log_2 N = x \iff 2^x = N$$

Questo valore cresce molto lentamente, per esempio $\log_2 10^6 \approx 20$.

A differenza della ricerca lineare, la ricerca binaria è applicabile anche su spazi di ricerca molto grandi.

Ricerca binaria

Implementazione

```
int l = 0, r = n+1; // [l, r)

while (r - l > 1) {
  int mid = (l + r) / 2;
  if (v[mid]==0) {
    l = mid;
  } else {
    r = mid;
  }
}

// return l;
```

Introduzione

Problema di esempio

Ritorniamo ora al problema precedente. Avevamo osservato che:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Introduzione

Problema di esempio

Ritorniamo ora al problema precedente. Avevamo osservato che:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Possiamo quindi immaginare un array in cui ogni elemento è 0 se B_x è valido, 1 altrimenti.

Ritorniamo ora al problema precedente. Avevamo osservato che:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Possiamo quindi immaginare un array in cui ogni elemento è 0 se B_x è valido, 1 altrimenti.
Questo array è della forma $[\dots, 0, 0, 1, 1, \dots]$.

Introduzione

Problema di esempio

Ritorniamo ora al problema precedente. Avevamo osservato che:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Possiamo quindi immaginare un array in cui ogni elemento è 0 se B_x è valido, 1 altrimenti.
Questo array è della forma $[\dots, 0, 0, 1, 1, \dots]$. Possiamo quindi utilizzare la ricerca binaria per trovare la risposta velocemente!

Introduzione

Problema di esempio

Ritorniamo ora al problema precedente. Avevamo osservato che:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Possiamo quindi immaginare un array in cui ogni elemento è 0 se B_x è valido, 1 altrimenti.
Questo array è della forma $[\dots, 0, 0, 1, 1, \dots]$. Possiamo quindi utilizzare la ricerca binaria per trovare la risposta velocemente!

- sappiamo controllare se un certo B è valido in $O(N)$

Introduzione

Problema di esempio

Ritorniamo ora al problema precedente. Avevamo osservato che:

- se B_x è valido, allora $B_y < B_x$ è anch'esso valido
- se B_x è non valido, allora $B_y > B_x$ è anch'esso non valido

Possiamo quindi immaginare un array in cui ogni elemento è 0 se B_x è valido, 1 altrimenti.
Questo array è della forma $[\dots, 0, 0, 1, 1, \dots]$. Possiamo quindi utilizzare la ricerca binaria per trovare la risposta velocemente!

- sappiamo controllare se un certo B è valido in $O(N)$
- possiamo usare la ricerca binaria per trovare il più grande B valido facendo $O(\log N)$ controlli.

La complessità totale è quindi $O(N \log N)$, che è sufficiente per entrare nel limite di tempo.

Qui potete testare le vostre soluzioni

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

Lotteria di quadri

Data una sequenza di $N \leq 200\,000$ *interi positivi e un intero* M*, trovare il massimo* $B \leq N$ *tale che la somma di ogni sottosegmento lungo* B *sia al più* M*.*

Problemi addizionali

https://training.olinfo.it/#/task/ois_tickets/statement
https://training.olinfo.it/#/task/ois_annoluce/statement