Competitive Programming

Corso per le Olimpiadi Italiane di Informatica

Creato da Giovanni Spadaccini

Overview del corso

1. **Prerequisiti, I/O e Complessità**
2. **Vettori e Greedy**
3. **Ricorsione**
4. **Programmazione Dinamica**
5. **Grafi**

Dividiamo il corso in moduli chiari e strutturati per una preparazione completa.

Risorse e Materiale di Studio

- Libro di riferimento: Guida Quinta Edizione
- Esercizi online per pratica e approfondimenti. OII
- Documentazione C++ e Python
- Di più alla fine

Lezione di oggi

- Tipi
- Input/Output
- Costrutti di base
- Funzioni
- Complessità

> Alcuni di voi sono più veterani e questi argomenti potrebbero essere già noti. In tal caso, vi propongo di aiutare i vostri compagni o di approfondire gli argomenti con esercizi più complessi (su cui posso dare consiglio solo alla fine della lezione).

Tipi di Dati in C++

Una panoramica dei tipi di dati fondamentali in C++ e la loro importanza.

Introduzione ai Tipi di Dati

- I tipi di dati definiscono la natura dei dati che una variabile può contenere.
- In C++, i tipi di dati sono categorizzati in:
  - Tipi fondamentali (es. int, char)
  - Tipi composti (es. array, classi)

Tipi Interi

- `int`: Numero intero, tipicamente di 4 byte.
- `short int`: Intero corto, due byte di un `int`.
- `long int`: Intero lungo, il doppio di un `int`.

Tipo Carattere

- `char`: Singolo carattere o numero piccolo (1 byte).

Tipi in Virgola Mobile

- `float`: Numero a virgola mobile, precisione singola.
- `double`: Numero a virgola mobile, precisione doppia.
- `long double`: Precisione maggiore di `double`.

Tipo Booleano

- `bool`: Rappresenta verità (`true`) o falsità (`false`).

I Tipi di Dati e la loro Importanza

- **Controllo del tipo**: Aiuta a prevenire errori, come dividere una stringa per un numero.
- **Efficienza**: Usare il tipo corretto può influenzare la memoria e la velocità del programma.
- **Leggibilità**: Rende il codice più facile da leggere e mantenere.

Tipi Composti

- **Array**: Collezione di elementi dello stesso tipo.
- **Struct**: Permette di combinare vari tipi di dati in un singolo tipo.
- **Pointer**: Variabile che contiene l'indirizzo di memoria di un'altra variabile.

Esercizio: Tipi e Operazioni

Dato un `int`, un `double` e un `char`, scrivere un programma che stampi la loro somma (nota: convertire tutti in `double`).

```
int a = 5;
double b = 6.2;
char c = '3'; // '3' è 51 in ASCII
```

Soluzione

```
double somma = a + b + (c - '0');
cout << "La somma è: " << somma << endl;
```

Input e Output

```cpp
// C++
#include <iostream>
using namespace std;

int main() {
    int a, b;
    cin >> a >> b;
    cout << "La somma è: " << a + b << endl;
}
```

```python
# Python
a = int(input())
b = int(input())
print("La somma è:", a + b)
```

Questa slide mostra come leggere due numeri interi e stampare la loro somma, sia in C++ che in Python.

Formattazione dell'Output

```cpp
// C++
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    double pi = 3.14159;
    cout << fixed << setprecision(2) << pi << endl; // "3.14"
}
```

```python
# Python
pi = 3.14159
print(f"{pi:.2f}")  # "3.14"
```

Esempi di come controllare la precisione della stampa di numeri in virgola mobile.

Lettura da un file

Non è il classico metodo di input, ma è veloce da scrivere.

```
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
```

```
import sys
sys.stdin = open("input.txt", "r")
sys.stdout = open("output.txt", "w")
```

Costrutti di Base

- `if`
- `while`
- `for`

Uso dell'istruzione `if`

Confronto tra Python e C++

Impareremo come utilizzare l'istruzione `if` per controllare le condizioni nei programmi.

Esempio di `if` in Python

```
x = 5
if x > 0:
    print("x è positivo")
else:
    print("x non è positivo")
```

Questo codice controlla se `x` è maggiore di zero e stampa un messaggio appropriato.

Esempio di `if` in C++

```cpp
#include <iostream>
using namespace std;

int main() {
    int x = 5;
    if (x > 0) {
        cout << "x è positivo" << endl;
    } else {
        cout << "x non è positivo" << endl;
    }
    return 0;
}
```

Simile all'esempio Python, questo codice in C++ stampa un messaggio basato sul valore di `x`.

Esercizio: Pari o Dispari

Condizione: Dato un numero intero, stampare `"Pari"` se il numero è pari, `"Dispari"` se il numero è dispari e `Negativo` se è minore di zero.

```python
numero = 4
if numero % 2 == 0:
    print("Pari")
elif numero< 0 :
    print("Negativo")
else:
    print("Dispari")
```

Con `numero = 4`, il programma stamperà `"Pari"`.

```cpp
#include <iostream>
using namespace std;

int main() {
    int numero = 4;
    if (numero % 2 == 0) {
        cout << "Pari" << endl;
    } else if (numero < 0) {
        cout << "Negativo" << endl;
    } else {
        cout << "Dispari" << endl;
    }
    return 0;
}
```

Anche qui, con `numero = 4`, il programma stamperà `"Pari"`.

Uso delle istruzioni `while` e `for`

Confronto tra Python e C++

Impareremo come utilizzare le istruzioni di ciclo `while` e `for` per iterare azioni nei programmi.

Esempio di `while` in Python

```
i = 0
while i < 5:
    print(f"i vale {i}")
    i += 1
```

Questo codice stampa i valori di `i` da 0 a 4.

Esempio di `while` in C++

```cpp
#include <iostream>
using namespace std;

int main() {
    int i = 0;
    while (i < 5) {
        cout << "i vale " << i << endl;
        i++;
    }
    return 0;
}
```

Simile all'esempio Python, questo codice in C++ stampa i valori di `i` da 0 a 4.

Esempio di `for` in Python

```
for i in range(5):
    print(f"i vale {i}")
```

Questo codice stampa i valori di `i` da 0 a 4, utilizzando un ciclo `for`.

Esempio di `for` in C++

```cpp
#include <iostream>
using namespace std;

int main() {
    for (int i = 0; i < 5; i++) {
        cout << "i vale " << i << endl;
    }
    return 0;
}
```

Anche qui, questo codice in C++ utilizza un ciclo `for` per stampare i valori di `i` da 0 a 4.

Esercizio: Somma dei numeri

Condizione: Calcolare la somma dei primi N numeri naturali, dove N è un intero dato.

Soluzione in Python

```python
N = 5  # Cambia questo valore per testare
somma = 0
for i in range(1, N + 1):
    somma += i
print(f"La somma è {somma}")
```

Con `N = 5`, il programma stamperà `"La somma è 15"`.

Soluzione in C++

```cpp
#include <iostream>
using namespace std;

int main() {
    int N = 5;  // Cambia questo valore per testare
    int somma = 0;
    for (int i = 1; i <= N; i++) {
        somma += i;
    }
    cout << "La somma è " << somma << endl;
    return 0;
}
```

Anche qui, con `N = 5`, il programma stamperà `"La somma è 15"`.

Array in C++

Gli array consentono di memorizzare più elementi dello stesso tipo in una singola variabile.

Definizione di un Array

Per definire un array in C++, si specifica il tipo di elemento, il nome dell'array e il numero di elementi che può contenere.

Sintassi

```
tipo nomeArray[dim];
```

Esempio

```
int numeri[5];
```

Questo dichiara un array di `int` che può contenere 5 elementi.

Inizializzazione di un Array

Gli array possono essere inizializzati al momento della dichiarazione.

Esempio

```
int numeri[5] = {0, 1, 2, 3, 4};
```

Se non tutti gli elementi sono specificati, gli elementi restanti vengono impostati a zero.

Accesso agli Elementi dell'Array

Accedi a un elemento dell'array utilizzando l'indice, che inizia da 0.

Esempio

```
int primoNumero = numeri[0]; // Accede al primo elemento
numeri[4] = 5; // Modifica il quinto elemento
```

Iterazione sugli Elementi di un Array

Utilizzare un ciclo `for` per iterare sugli elementi di un array.

Esempio

```
for(int i = 0; i < 5; i++) {
    cout << numeri[i] << endl;
}
```

Questo stampa tutti gli elementi dell'array `numeri`.

Limitazioni degli Array in C++

- La dimensione dell'array deve essere nota al momento della compilazione e non può essere modificata dinamicamente.
- Gli array non mantengono informazioni sulla loro dimensione.

Per superare queste limitazioni, C++ offre tipi di dati più avanzati, come i vettori (`std::vector`).

Array Multidimensionali

Gli array possono avere più dimensioni, ad esempio, per rappresentare una matrice.

Esempio

```
int matrice[2][3] = {
  {1, 2, 3},
  {4, 5, 6}
};
```

Esercizio: Calcolo della Media

Dato un array di numeri interi, calcolare la media dei suoi elementi.

Soluzione

```
int numeri[5] = {1, 2, 3, 4, 6};
int somma = 0;
for(int i = 0; i < 5; i++) {
    somma += numeri[i];
}
double media = ((double) somma) / 5;
cout << "La media è: " << media << endl;
```

Liste in Python

Le liste in Python sono strutture dati simili agli array, ma più flessibili.

Definizione di una Lista

Una lista può contenere elementi di diversi tipi e la sua dimensione può variare.

Sintassi

```
nomeLista = [elemento1, elemento2, ...]
```

Esempio

```
numeri = [1, 2, 3, 4, 5]
```

Questo crea una lista di `int` con 5 elementi.

Accesso agli Elementi della Lista

Accedi a un elemento della lista utilizzando l'indice, che inizia da 0.

Esempio

```
primoNumero = numeri[0] # Accede al primo elemento
numeri[4] = 6 # Modifica il quinto elemento
```

Aggiungere e Rimuovere Elementi

Python permette di modificare facilmente le liste.

Aggiungere Elementi

```
numeri.append(6) # Aggiunge un elemento in fondo alla lista
```

Rimuovere Elementi

```
numeri.remove(2) # Rimuove il primo elemento con valore 2
```

Iterazione sugli Elementi di una Lista

Utilizzare un ciclo `for` per iterare sugli elementi di una lista.

Esempio

```
for numero in numeri:
    print(numero)
```

Questo stampa tutti gli elementi della lista `numeri`.

Esercizio: Calcolo della Media

Dato una lista di numeri interi, calcolare la media dei suoi elementi.

Soluzione

```
numeri = [1, 2, 3, 4, 5]
media = sum(numeri) / len(numeri)
print(f"La media è {media}")

# for i in range(len(numeri)):
#     somma += numeri[i]
# media = somma / len(numeri)
```

Funzioni in C++ e Python

Le funzioni sono blocchi di codice riutilizzabili progettati per eseguire una specifica operazione.

Definizione di una Funzione

Una funzione è definita specificando il tipo di ritorno, il nome della funzione, e parametri tra parentesi.

In C++

```
tipoRitorno nomeFunzione(parametro1, parametro2) {
    // Corpo della funzione
}
```

In Python

```
def nomeFunzione(parametro1, parametro2):
    # Corpo della funzione
```

Esempio di Funzione

Creiamo una funzione per sommare due numeri.

In C++

```cpp
#include <iostream>
using namespace std;

int somma(int a, int b) {
    return a + b;
}

int main() {
    cout << somma(5, 3); // Stampa 8
}
```

In Python

```python
def somma(a, b):
    return a + b

print(somma(5, 3)) # Stampa 8
```

Parametri e Argomenti

- I **parametri** sono le variabili specificate nella definizione della funzione.
- Gli **argomenti** sono i valori forniti quando la funzione è chiamata.

Funzioni Senza Ritorno

Le funzioni possono non restituire alcun valore.

In C++

```cpp
void saluta() {
    cout << "Ciao a tutti!" << endl;
}
```

In Python

```python
def saluta():
    print("Ciao a tutti!")
```

Valori di Ritorno

Le funzioni possono restituire valori al codice che le ha chiamate usando `return`.

Esempio

In C++ e Python, `return valore;` restituisce il valore al chiamante.

Parametri Predefiniti

Si possono definire valori predefiniti per i parametri delle funzioni.

In C++

```cpp
int somma(int a, int b = 5) {
    return a + b;
}
```

In Python

```python
def somma(a, b=5):
    return a + b
```

Esercizio: Calcolo del Fattoriale

Scrivi una funzione che calcoli il fattoriale di un numero.

Soluzione in C++

```cpp
int fattoriale(int n) {
    if (n == 0) return 1;
    return n * fattoriale(n - 1);
}
```

Soluzione in Python

```python
def fattoriale(n):
    if n == 0:
        return 1
    return n * fattoriale(n - 1)
```

Esercizio testo

Abbiamo N sticks con cui dobbiamo creare un rettangolo con area massima.

Input:

- N: numero di sticks
- N interi: lunghezza dei bastoncini

Output:

- Area massima del rettangolo (zero se non è possibile formare un rettangolo)

Input e output

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    int n;
    cin >>n;
    long long  arr[n];
    for(int i=0;i<n;i++){
        cin >>arr[i];
    }
    int first=-1,second=-1;
    if(second==-1){
        cout <<0<<endl;
    }else{
        cout << first*second;
    }
}
```

```python
# Riceve il numero di elementi da input
n = int(input())

# Riceve gli elementi e li inserisce in una lista
arr = []
for _ in range(n):
    arr.append(int(input()))

first = -1
second = -1
if second == -1:
    print(0)
print(first * second)

exit(0)
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    int n;
    cin >>n;
    long long  arr[n];
    for(int i=0;i<n;i++){
        cin >>arr[i];
    }
    sort(arr,arr+n);
    long long first=-1,second=-1;
    long long last = arr[n-1];
    for(int i=n-2;i>0;i--){
        if(arr[i]==last){
            if(first==-1){
                first=arr[i];
            }else{
                second=arr[i];
                break;
            }
        }
        last=arr[i];
    }
    if(second==-1){
        cout <<0<<endl;
    }
    cout << first*second;

}
```

```python
# Riceve il numero di elementi da input
n = int(input())

# Riceve gli elementi e li inserisce in una lista
arr = []
for _ in range(n):
    arr.append(int(input()))

# Ordina la lista
arr.sort()

# Inizializza le variabili per il primo e il secondo elemento più grande
first = -1
second = -1

# Imposta l'ultimo elemento come l'elemento più grande
last = arr[-1]

# Cerca il primo e il secondo elemento più grande
for i in range(n-2, 0, -1): # Itera al contrario partendo dal penultimo elemento
    if arr[i] == last:
        if first == -1:
            first = arr[i]
        else:
            second = arr[i]
            break
    last = arr[i]

# Se non è stato trovato un secondo elemento, stampa 0,
# altrimenti stampa il prodotto del primo e del secondo elemento più grande
if second == -1:
    print(0)
else:
    print(first * second)

exit(0)
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    int n;
    cin >>n;
    long long  arr[n];
    for(int i=0;i<n;i++){
        cin >>arr[i];
    }
    sort(arr,arr+n);
    long long first=-1,second=-1;
    long long last = arr[n-1];
    for(int i=n-2;i>=0;i--){
        if(arr[i]==last){
            if(first==-1){
                first=arr[i];
                i--;
            }else{
                second=arr[i];
                break;
            }
        }
        last=arr[i];
    }
    if(second==-1){
        cout <<0<<endl;
    }else{
        cout << first*second;
    }
}
```

```python
# Riceve il numero di elementi da input
n = int(input())

# Riceve gli elementi e li inserisce in una lista
arr = []
for _ in range(n):
    arr.append(int(input()))

# Ordina la lista
arr.sort()

# Inizializza le variabili per il primo e il secondo elemento più grande
first = -1
second = -1

# Imposta l'ultimo elemento come l'elemento più grande
last = arr[-1]

# Cerca il primo e il secondo elemento più grande
for i in range(n-2, -1, -1): # Itera al contrario partendo dal penultimo elemento
    if arr[i] == last:
        if first == -1:
            first = arr[i]
            i -= 1
        else:
            second = arr[i]
            break
    last = arr[i]

# Se non è stato trovato un secondo elemento, stampa 0,
# altrimenti stampa il prodotto del primo e del secondo elemento più grande
if second == -1:
    print(0)
else:
    print(first * second)

exit(0)
```

Introduzione

In competitive programming, l'efficienza e la rapidità nello scrivere codice sono cruciali. Utilizzare codice preesistente tramite `import` in Python e `#include` in C++ può aiutarti a concentrarti sulla soluzione del problema.

`#include` in C++

- `#include` consente di includere il contenuto di un file sorgente o di una libreria standard nel punto in cui appare la direttiva.
- Molto usato per includere librerie standard come `<iostream>` per l'input/output o `<vector>` per utilizzare i vettori.

Esempio

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> v = {1, 2, 3};
    std::cout << v[0]; // Stampa 1
}
```

`import` in Python

- `import` carica un modulo o una libreria, rendendo le sue funzioni e classi disponibili nel tuo script.
- Fondamentale per accedere a moduli standard come `math` o a moduli di terze parti.

Esempio

```
import math

print(math.sqrt(4)) # Stampa 2.0
```

Utilizzo in Competitive Programming

C++

- Utilizzare `#include` per accedere a strutture dati e algoritmi efficaci, come `std::sort` o `std::set`.
- `#include <bits/stdc++.h>` include quasi tutte le librerie standard (usato comunemente in competitive programming, ma da evitare in produzione).

Python

- `import` è utile per moduli come `math`, `itertools` o `sys` per leggere velocemente da stdin.
- Python ha molte librerie per facilitare la manipolazione di dati e calcoli matematici.

Vantaggi

- **Riutilizzo del Codice:** Evita di riscrivere codice comune, concentrandoti sulla logica specifica del problema.
- **Efficienza:** Sfrutta implementazioni ottimizzate di strutture dati e algoritmi.
- **Leggibilità:** Il codice è più ordinato e facile da comprendere.

Considerazioni Finali

- Sia `#include` in C++ che `import` in Python sono strumenti potenti per il competitive programming.
- Imparare quali librerie sono disponibili e come utilizzarle può significativamente migliorare la qualità e l'efficienza delle tue soluzioni.

Complessità Computazionale

Esploriamo come misurare e comprendere l'efficienza dei nostri programmi.

Cos'è la Complessità Computazionale?

- **Complessità Computazionale** ci dice quanto è "costoso" eseguire un algoritmo in termini di tempo (quanto è lungo) e spazio (quanta memoria usa).
- È come confrontare quale macchina è più veloce o quale consuma meno benzina.

Tipi di Complessità

- **Complessità Temporale:** Quanto tempo impiega un algoritmo.
- **Complessità Spaziale:** Quanta memoria occupa un algoritmo.

Complessità Temporale: Esempio

Ricerca Lineare

- Cerca un elemento in un array scorrendolo dall'inizio alla fine.
- Complessità: O(n), dove n è il numero di elementi.

Ricerca Lineare in C++

```cpp
#include <iostream>
using namespace std;

bool ricercaLineare(int arr[], int n, int x) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) return true;
    }
    return false;
}
```

Ricerca Lineare in Python

```
def ricerca_lineare(arr, x):
    for elemento in arr:
        if elemento == x:
            return True
    return False
```

Altri Esempi di Complessità Computazionale

Esploriamo la complessità O(n^2) e O(2^n) con esempi pratici.

Complessità O(n^2): Bubble Sort

Bubble sort confronta ripetutamente coppie di elementi adiacenti e li scambia se sono in ordine sbagliato. Ha una complessità di O(n^2).

```cpp
#include <iostream>
using namespace std;

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}
```

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

Complessità O(2^n): Fibonacci Ricorsivo

La sequenza di Fibonacci calcolata ricorsivamente ha una complessità temporale di O(2^n) a causa delle molteplici ripetizioni di calcoli per lo stesso valore.

```cpp
#include <iostream>
using namespace std;

int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

```python
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Impatto della Complessità

- **O(n^2)**: Cresce rapidamente con l'aumentare di n, diventando inefficiente per grandi dati.
- **O(2^n)**: Cresce ancora più rapidamente, rendendo l'algoritmo praticamente inutilizzabile per grandi valori di n.

Considerazioni

- La scelta degli algoritmi è cruciale per la gestione efficiente dei dati.
- Algoritmi con migliore complessità computazionale possono gestire dati più grandi più velocemente.

Complessità Temporale: Esempio

Ricerca Binaria

- Cerca un elemento in un array ordinato dividendo l'array a metà ad ogni passo.
- Complessità: O(log n), molto più veloce su grandi quantità di dati.

Ricerca Binaria in C++

```cpp
#include <iostream>
using namespace std;

bool ricercaBinaria(int arr[], int l, int r, int x) {
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (arr[m] == x) return true;
        if (arr[m] < x) l = m + 1;
        else r = m - 1;
    }
    return false;
}
```

Ricerca Binaria in Python

```python
def ricerca_binaria(arr, x):
    l, r = 0, len(arr) - 1
    while l <= r:
        m = (l + r) // 2
        if arr[m] == x:
            return True
        elif arr[m] < x:
            l = m + 1
        else:
            r = m - 1
    return False
```

Perché è Importante?

- **Efficienza:** Programmi più veloci e meno consumatori di risorse.
- **Scalabilità:** Gestire grandi quantità di dati senza rallentamenti.

Esercizio Pratico

Confrontare il tempo di esecuzione della ricerca lineare e quella binaria su un array di 1 milione di numeri.

- Prevedete quale sarà più veloce?
- Come potete testare e misurare il tempo effettivo?

Struttura delle gare

Materiali per approfondimenti

- Libro di riferimento: Guida Quinta Edizione
- Lezioni tenute da ragazzi di codefarm: https://www.youtube.com/playlist?list=PLxSVZC2doc7cxBBLqoMlCjOd6MeMFRnbl
- Se volte sperimentare con i problemi che troverete alla gara https://territoriali.olinfo.it/

Materiali avanzati (livello nazionali)

molto comprensiva di materiali (anzi troppo)

- Buon modo per iniziare e guardare diversi argomenti: https://algobadge.olinfo.it/
- Libro di riferimento gratis (in inglese): https://cses.fi/book/book.pdf
- Tutti le implementazioni di tutti i più svariati algoritmi: https://cp-algorithms.com/
- Principale sito per i Competitive Programmers codeforces.com
- Buon modo per iniziare anche se molto pi`u complicato : https://usaco.guide/
- Corso avanzato codefarm: https://www.youtube.com/playlist?list=PLxSVZC2doc7cxBBLqoMlCjOd6MeMFRnbl

Problemi da provare

- tamburello https://training.olinfo.it/#/task/tamburello
- scommessa https://training.olinfo.it/#/task/scommessa/statement
- collatz https://training.olinfo.it/#/task/collatz/
- Crittografia LWF https://training.olinfo.it/#/task/lwf/submissions

Più complicati (nozioni non spiegate in questa lezione)

- quasi palindromo https://training.olinfo.it/#/task/quasipal
- sbarramento https://training.olinfo.it/#/task/sbarramento