Progetto “Grafo delle chiamate tra funzioni”

Appello dell’8 febbraio 2021

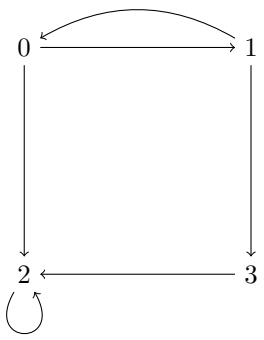
Descrizione del progetto

Lo scopo del progetto è quello di analizzare un grafo rappresentante le chiamate tra diverse funzioni che compongono un programma e individuare:

- Se vi siano funzioni ricorsive. Ovvero se una funzione `f` chiama sé stessa.
- Se vi siano coppie di funzioni mutualmente ricorsive. Ovvero se due funzioni `f` e `g` sono tali per cui `f` chiama `g` e `g` chiama `f`.
- Se, date due funzioni `f` e `g`, la funzione `g` viene chiamata da `f` direttamente o indirettamente (e.g., `f` chiama `h` che chiama `g`).
- Se vi siano funzioni mai chiamate a partire dalla funzione `main`.

Per semplicità, in un grafo di chiamate con n funzioni le funzioni sono identificate dai numeri da 0 a $n-1$. Il grafo è fornito come lista di adiacenza.

Per esempio si consideri il seguente grafo in cui sono evidenziate tramite archi le chiamate tra 4 funzioni (numerate da 0 a 3):



Esso verrà fornito in input come la seguente lista di liste: `[[1, 2], [0, 3], [2], [2]]`.

Si implementi una classe `GrafoChiamate` con i seguenti metodi:

- `__init__(self, grafo)`. Il costruttore riceve come argomento un grafo nel formato specificato sopra (lista di adiacenza).
- `funzioni_ricorsive(self)`. Questo metodo ritorna un array (lista python) di tutte le funzioni ricorsive, ovvero che, nel grafo hanno arco che forma un cappio (le due estremità coincidono). Nel grafo dell’esempio l’unica funzione ricorsiva è la `2`.
- `funzioni_mutualmente_ricorsive(self)`. Questo metodo ritorna un array (lista python) di tutte le funzioni mutualmente ricorsive. Nel grafo dell’esempio queste sono le due funzioni `0` e `1`, in quanto `0` chiama `1` che chiama `0`.

- `chiama(self, f, g)`. Questo metodo riceve come argomenti due numeri interi rappresentanti le funzioni e ritorna un valore Booleano che è `True` se, partendo da `f`, è possibile effettuare una sequenza di chiamate di funzione che arriva a `g`. Per esempio con argomenti `0` e `3` il valore ritornato sarà `True` (`0` chiama `1` che chiama `3`), mentre con argomenti `2` e `3` il valore ritornato sarà `False`.
- `dead_code(self, f)`. Questo metodo ritorna un array (lista python) contenente tutte le funzioni non raggiungibili a partire da `f`. Per esempio con argomento `0` il valore ritornato sarà `[]`, mentre con `2` l'array di funzioni non chiamabili a partire `2` è `[0, 1, 3]`.

Si richiede di utilizzare l'algoritmo di Floyd–Warshall o una variante di esso all'interno della classe `GrafoChiamate` e convertire il grafo in una matrice di adiacenza per l'utilizzo di quell'algoritmo.

Codice

Viene fornito uno scheletro del codice che deve essere implementato, con i metodi che devono essere scritti:

```
class GrafoChiamate:

  def __init__(self, grafo):
    #...

  def funzioni_ricorsive(self):
    #...

  def funzioni_mutualmente_ricorsive(self):
    #...

  def chiama(self, f, f):
    #...

  def dead_code(self, f):
    #...
```

Nel progetto è consentito avere funzioni e classi aggiuntive per l'implementazione. Per esempio, per calcolare la distanza (in termini di numero di chiamate) tra tutte le coppie di funzioni.

Si ricorda di commentare adeguatamente il codice, approfittandone per spiegare le scelte implementative effettuate.

Esempi d'uso

Il seguente frammento di codice chiama tutti i metodi la cui implementazione è richiesta dal progetto:

```python
def test():
    g = GrafoChiamate([[1, 2], [0, 3], [2], [2]])
    print(g.funzioni_ricorsive())
    print(g.funzioni_mutualmente_ricorsive())
    print(g.chiama(0, 3))
    print(g.chiama(3, 1))
    print(g.dead_code(0))
    print(g.dead_code(2))
```

Questo dovrebbe essere l'output prodotto da una implementazione funzionante:

```
[2]
[0, 1]
True
False
[]
[0, 1, 3]
```

Suggerimenti generali

1. Sebbene per risolvere lo stesso problema possano esistere altre strutture dati, per l'esame è da implementare la struttura dati descritta nel testo del progetto.
2. Rispettate i vincoli di complessità richiesti.
3. Fate più test possibile: assicuratevi che il codice implementato funzioni in tutti i casi richiesti.

Indicazioni

Il progetto deve essere svolto **individualmente**. La consegna dovrà avvenire entro le ore 23:59 del giorno 29/01/2021 secondo le seguenti modalità:

- Invio di una email a `lmanzoni@units.it` dal vostro account email istituzionale con oggetto *[informatica] consegna progetto appello del 08/02/2021*.
- L'email deve avere come allegato il progetto in un singolo file in codice sorgente Python versione 3, dal nome `Nome_Cognome_matricola.py`, quindi, per esempio Mario Rossi di matricola 12345 consegnerà un file dal nome `Mario_Rossi_12345.py`.
- Il file deve contenere sotto forma di commento le seguenti linee indicanti nome, cognome e numero di matricola:

  ```python
  # Nome: Mario
  # Cognome: Rossi
  # Matricola: 12345
  ```