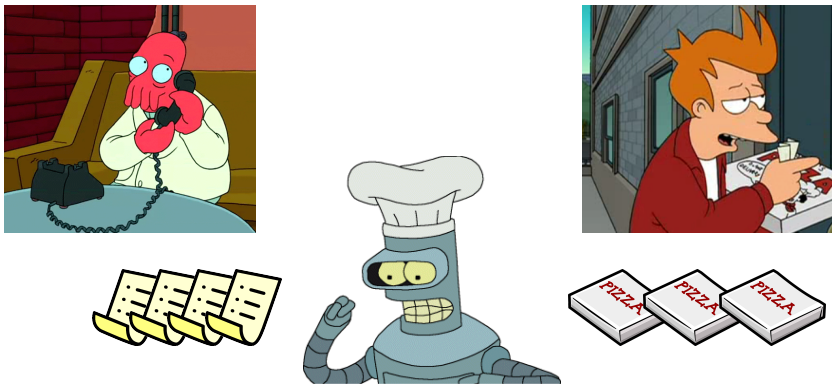
Lezione 4 – Le code

Informatica

26 Aprile 2016

Bender il pizzaiolo robot



Bender ha aperto una pizzeria da asporto con i suoi amici:

- Dr. Zoidberg risponde al telefono e riceve gli ordini
- Bender prepara le pizze
- Fry le consegna

Implementiamo `bender`

Scriviamo delle funzioni che simulano il comportamento di Bender:

1. una funzione che aggiunge un nuovo ordine all'elenco
2. una funzione che estrae un ordine dall'elenco, prepara la pizza corrispondente e la mette tra quelle pronte per la consegna
3. una funzione che estrae la prossima pizza da consegnare

Le pizze devono essere preparate e consegnate seguendo l'ordine di arrivo degli ordini

Ordini e pizze: due esempi di coda

Gli ordini in attesa e le pizze pronte per la consegna sono due esempi di *code*:

- sono tipi di dati *mutabili*:
  - nuovi elementi vengono inseriti, vecchi elementi vengono tolti
- I nuovi elementi vengono sempre inseriti *alla fine* della coda
- Gli elementi da togliere vengono estratti *dall'inizio* della coda
- Questa politica di inserimento/cancellazione si chiama *FIFO* (First-In, First-Out)

Le code: proprietà e operazioni

Una coda (in inglese *queue*, pronunciato *"chiu"*) ha le seguenti proprietà:

- è una collezione di elementi
- mantiene l'ordine degli elementi che contiene
- è dotata di un inizio (*head*) e di una fine (*tail*)

Le operazioni principali che posso eseguire su una coda sono:

- *Accodamento di un elemento*: Detta anche operazione di *enqueue*, aggiunge un elemento alla fine della coda.
- *Estrazione di un elemento*: Detta anche operazione di *dequeue*, rimuove un elemento dall'inizio della coda.

Tipi di Dati Astratti

Un *tipo di dato astratto* è un tipo di dato *definito unicamente* da:

- i *valori* che può assumere
- le *proprietà* che deve rispettare
- le *operazioni* che si possono eseguire sul tipo di dato
- il *comportamento* (o *semantica*) delle operazioni

I tipi di dati astratti distinguono nettamente tra:

- *interfaccia*: le operazioni che vengono fornite per la manipolazione del dato
- *implementazione*: il modo in cui sono rappresentati i dati ed eseguite le operazioni

L'implementazione viene considerata *inaccessibile* all'utente, che può operare sul dato solo mediante l'interfaccia (*incapsulamento*)

Perché i tipi di dati astratti?

Algoritmi + Strutture Dati = Programmi
Niklaus Wirth, 1976

- Usare tipi di dati astratti vuol dire distinguere tra:
  - *algoritmi*: descrizioni ad alto livello delle operazioni da compiere
  - *strutture dati* che descrivono il modo in cui l'informazione è organizzata e manipolata
- Permettono di scrivere programmi più semplici e leggibili
  - non occorre conoscere i dettagli implementativi per capire cosa succede
- Facilitano l'identificazione e la correzione degli errori
- Se l'implementazione viene cambiata, non è necessario riscrivere i programmi:
  - l'interfaccia è sempre la stessa!

Le code: interfaccia

Creazione di nuove istanze

`queue()` Crea una coda vuota

Manipolazione

`enqueue(a, c)` Accoda `a` alla fine della coda `c`

`dequeue(c)` Rimuove il primo elemento della coda `c` e ne ritorna il valore

Controllo/Interrogazione

`is_empty(c)` Ritorna `True` se la coda `c` è vuota

`head(c)` Ritorna il primo elemento della coda `c` (senza rimuoverlo)

`tail(c)` Ritorna l'ultimo elemento della coda `c` (senza rimuoverlo)

`length(c)` Ritorna il numero di elementi nella coda `c`

Utilità

`print_queue(c)` Stampa il contenuto della coda `c`

Ora che conosciamo l'interfaccia del tipo di dato astratto "coda" possiamo scrivere il codice delle funzioni che implementano `bender` *senza dover conoscere l'implementazione*.

La funzione che inserisce un nuovo ordine modifica la coda `ordini`:

```
def nuovo_ordine(pizza, ordini):
    print("Zoidberg ha ricevuto un ordine per una", pizza)
    enqueue(pizza, ordini)
```

La funzione che prepara le pizze ha due argomenti: la coda degli ordini in attesa e la coda delle pizze pronte per la consegna.

```
def bender(ordini,pronte):
    if not is_empty(ordini):
        pizza = dequeue(ordini)
        print("Bender sta preparando una", pizza)
        enqueue(pizza, pronte)
    else:
        print("Bender non ha pizze da preparare")
```

Infine, la funzione che prepara le consegne opera solo sulla coda delle pizze pronte.

```
def consegna(pronte):
    if not is_empty(pronte):
        pizza = dequeue(pronte)
        print("Fry sta consegnando una", pizza)
    else:
        print("Fry non ha pizze da consegnare")

print(" --- BENDER IL PIZZAIOLO --- ")
ordini = queue()
pronte = queue()
nuovo_ordine("margherita", ordini)
nuovo_ordine("quattro stagioni", ordini)
print("ordini:", ordini)
```

```
print("pronte:", pronte)
bender(ordini, pronte)
print("ordini:", ordini)
print("pronte:", pronte)
bender(ordini, pronte)
print("ordini:", ordini)
print("pronte:", pronte)
consegna(pronte)
bender(ordini, pronte)
print("ordini:", ordini)
print("pronte:", pronte)
consegna(pronte)
consegna(pronte)
```

Le code: implementazione

Per ottenere un programma completo dobbiamo aggiungere l'implementazione delle funzioni che compongono l'interfaccia. Quello che segue è un esempio di implementazione che usa le liste.

```
def queue():
    return []

def enqueue(a, c):
    c.append(a)
    return c

def dequeue(c):
    if len(c) > 0:
        a = c[0]
        del c[0]
        return a
    return None

def head(c):
    if len(c) > 0:
        return c[0]
    else:
        return None

def tail(c):
    if len(c) > 0:
        return c[-1]
    else:
        return None

def is_empty(c):
    return len(c) == 0

def length(c):
    return len(c)

def print_queue(c):
    print(c)
```

Esercizi sulle code

In tutti gli esercizi che seguono le operazioni sulle code devono essere svolte usando solamente le funzioni dell'interfaccia descritte in precedenza.

Il pronto soccorso

Il pronto soccorso di un ospedale accetta e la cura i pazienti nel modo seguente:

- Ad ogni paziente viene assegnato un codice di priorità:
  - codice rosso per i casi gravi, e codice verde per i casi lievi
- Se c'è un medico disponibile, al medico viene assegnato un paziente secondo le seguenti regole:
  - i pazienti con codice rosso hanno la precedenza
  - se due pazienti hanno lo stesso codice, ha la precedenza chi è in attesa da più tempo

Scrivere delle funzioni per gestire l'accettazione di un paziente e l'assegnazione di un nuovo paziente ad un medico.

```
def accetta(paziente, codice, verdi, rossi):
    if codice == 'rosso':
        enqueue(paziente, rossi)
    elif codice == 'verde':
        enqueue(paziente, verdi)
    else:
        print("Codice di priorità non valido!")

def assegna(medico, verdi, rossi):
    if not is_empty(rossi):
        paziente = dequeue(rossi)
        return (medico, paziente)
    elif not is_empty(verdi):
        paziente = dequeue(verdi)
        return (medico, paziente)
    else:
        print("Nessun paziente in attesa")
        return (medico, None)

print(" --- IL PRONTO SOCCORSO --- ")
rossi = queue()
verdi = queue()
accetta('Mario', 'verde', verdi, rossi)
accetta('Elisa', 'verde', verdi, rossi)
p = assegna('Cox', verdi, rossi)
print(p[1], "assegnato al Dr.", p[0])
accetta('Luigi', 'rosso', verdi, rossi)
accetta('Pietro', 'verde', verdi, rossi)
p = assegna('Dorian', verdi, rossi)
print(p[1], "assegnato al Dr.", p[0])
p = assegna('Kelso', verdi, rossi)
print(p[1], "assegnato al Dr.", p[0])
```

Il problema di Giuseppe

"Guerra Giudaica" Flavio Giuseppe, 93–94 d.C.

Ci sono n persone disposte in cerchio in attesa di esecuzione. Scelta una persona iniziale e un senso di rotazione:

- si saltano $k-1$ persone
- la k-esima persona viene giustiziata ed eliminata dal cerchio;
- si riprende la conta saltando $k-1$ persone e si giustizia la k-esima persona;
- le esecuzioni proseguono e il cerchio si restringe sempre più, finché non rimane una sola persona, che viene graziata.

Dati n e k, determinare la posizione del sopravvissuto all'interno del cerchio iniziale.

```
def josephus(n, k):
    c = queue()
    for i in range(1,n+1):
        enqueue(i, c)
    p = 0
    while not is_empty(c):
        for i in range(1, k):
            p = dequeue(c)
            enqueue(p, c)
        p = dequeue(c)
    return p

print(" --- IL PROBLEMA DI GIUSEPPE --- ")
print(josephus(40,7))
```

Usando la funzione `length(c)` che ritorna il numero di elementi in coda si può risolvere il problema in questo modo:

```
def josephus(n, k):
    c = queue()
    for i in range(1,n+1):
        enqueue(i, c)
    while length(c) > 1:
        for i in range(1, k):
            p = dequeue(c)
            enqueue(p, c)
        dequeue(c)
    return head(c)
```

Invertire una coda

Scrivere una funzione ricorsiva che inverte l'ordine degli elementi di una coda. La funzione deve operare "in place":

- deve modificare la coda in ingresso e non crearne una nuova
- non può usare code, liste o altre strutture dati mutabili ausiliarie
- può usare variabili immutabili (p.es. per memorizzare singoli elementi della coda)

```
def invert_queue(c):
    if is_empty(c):
        return
    a = dequeue(c)
    invert_queue(c)
    enqueue(a, c)

print(" --- INVERTI CODA --- ")
q = enqueue(5, enqueue(4, enqueue(3, enqueue(2, enqueue(1, queue())))))
print_queue(q)
invert_queue(q)
print_queue(q)
```