

Lezione 5 – Le pile

Informatica

29 Aprile 2015

Il porto di Anversa

Il terminal del porto merci di Anversa usa delle “gru a cavaliere” per movimentare i container:

- I container arrivano per nave e vengono messi nel deposito, in attesa di essere spediti via camion alla loro destinazione finale
- Per risparmiare spazio, i container vengono impilati uno sopra l'altro, fino ad un'altezza massima di tre



Implementiamo carrier

Vogliamo scrivere una funzione che simula il comportamento delle gru all'arrivo di un camion. La gru deve:

1. identificare la posizione del container da caricare sul camion
2. se il container è in cima alla pila, può essere preso e caricato direttamente sul camion
3. se si trova sotto, si devono prima spostare i container che stanno sopra
4. alla fine dell'operazione i container spostati vanno rimessi nella pila di origine

Possiamo usare una coda per risolvere il problema?

Le pile: proprietà e operazioni

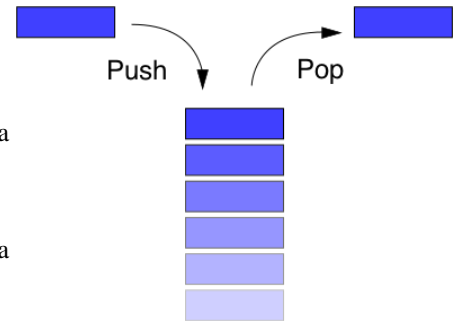
Una pila (in inglese *stack*) ha le seguenti proprietà:

- è una collezione di elementi
- mantiene l'ordine degli elementi che contiene
- è dotata di una cima (*top*)

Le operazioni eseguibili su una pila sono:

- *Impilamento di un elemento:*
Detta anche operazione di *push*, aggiunge un elemento in cima alla pila.
- *Estrazione di un elemento:*
Detta anche operazione di *pop*, rimuove l'elemento in cima alla pila.

Questa politica di inserimento/cancellazione si chiama *LIFO* (Last-In, First-Out)



Le pile: interfaccia

Creazione di nuove istanze

`stack()` Crea una pila vuota

Manipolazione

`push(a, p)` Aggiunge `a` in cima alla pila `p`

`pop(p)` Rimuove l'elemento in cima alla pila `p` e ne ritorna il valore

Controllo/Interrogazione

`is_empty(p)` Ritorna `True` se la pila `p` è vuota

`top(p)` Ritorna l'elemento in cima alla pila `p` (senza rimuoverlo)

Utilità

`print_stack(p)` Stampa il contenuto della pila `p`

Ora che conosciamo l'interfaccia del tipo di dato astratto "pila" possiamo scrivere il codice della funzione `carrier` senza dover conoscere l'implementazione. La funzione ha tre argomenti: la pila dei container, il codice del container ed il camion di destinazione.

```
def carrier(deposito, codice, camion):
    temp = stack()
    trovato = False
    while (not is_empty(deposito)) and (not trovato):
        container = pop(deposito)
        if container == codice:
            push(container, camion)
            trovato = True
        else:
            push(container, temp)
    while not is_empty(temp):
        container = pop(temp)
        push(container, deposito)

print(" --- IL PORTO DI ANVERSA --- ")
deposito=["AAA", "B3D", "QXF"]
camion = stack()
print_stack(deposito)
carrier(deposito, "AAA", camion)
print_stack(deposito)
```

```
print_stack(camion)
camion = stack()
carrier(deposito, "QXF", camion)
print_stack(deposito)
print_stack(camion)
```

Le pile: 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 stack():
    return []

def pop(p):
    if not is_empty(p):
        a = p[-1]
        del p[-1]
        return a
    return None

def push(a,p):
    p.append(a)

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

def top(p):
    if not (is_empty(p)):
        return(p[-1])
    else:
        return None

def print_stack(p):
    print("-->")
    for i in range(len(p)-1,-1,-1):
        print(p[i])
    print("----")
```

Esercizi sulle pile

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

Invertire e copiare pile

1. Scrivere una funzione che prende come argomento una pila e restituisce una nuova pila che contiene i valori della pila di input in ordine inverso.
2. Scrivere una funzione che crea una copia della pila passata in ingresso.

```
def inverti(p):
    q = stack()
    r = stack()
    while not is_empty(p):
        a = pop(p)
        push(a, q)
        push(a, r)
    while not is_empty(r):
        push(pop(r), p)
    return q

def copia(p):
    q = stack()
    r = stack()
    while not is_empty(p):
        a = pop(p)
        push(a, q)
    while not is_empty(q):
        a = pop(q)
        push(a, r)
        push(a, p)
    return r

print(" --- COPIA E INVERTI --- ")
p = stack()
push(1, p)
push(2, p)
push(3, p)
q = inverti(p)
print_stack(q)
q = copia(p)
print_stack(q)
```

Il bilanciamento delle parentesi

Scrivere una funzione per verificare che le parentesi tonde e quadre all'interno di una stringa siano correttamente annidate.

Esempi:

- [()] è bilanciata
- [()] () non è bilanciata
- [eke(jfj)jeje] è bilanciata
- {eke[jfj]je} non è bilanciata

```
def balanced(s):
    p=stack()
    for c in s:
        if c in ['(', '[', '{']:
            push(c,p)
        if c in [')', ']', '}']:
            if is_empty(p):
                return False
            d = pop(p)
            if not ((d,c) in [( '(', ')'), ('[', ']'), ('{', '}')]):
                return False
    return is_empty(p)

print(" --- BALANCED --- ")
print(balanced('[ ( ) ]'))
print(balanced('[eke(jfj)jeje]'))
print(balanced('{eke[jfj]je}'))
```

Il labirinto

Realizzare una funzione che, dato un labirinto $n \times n$, restituisca True se è possibile raggiungere l'ultima cella in basso a destra (in posizione $n - 1, n - 1$) partendo dalla prima cella in alto a sinistra (posizione 0, 0) passando solo per celle del labirinto di valore 0 (le celle di valore 1 rappresentano i muri del labirinto). Utilizzare una pila.

```
labirinto = [
    [ 0, 0, 0, 1, 0 ],
    [ 0, 1, 1, 1, 0 ],
    [ 0, 0, 0, 1, 1 ],
    [ 0, 1, 0, 0, 0 ],
    [ 0, 0, 1, 1, 0 ] ]
```

I numeri in rosso rappresentano un possibile cammino.

```
def risolvi(lab):
    n = len(lab)
    p = stack()
    push((0,0), p)
    while not is_empty(p):
        (i,j) = top(p)
        if (i,j) == (n-1, n-1):
            return True
        # marca la casella come visitata
        lab[i][j] = 2
        # vai su
        if i > 0 and lab[i-1][j] == 0:
            push((i-1,j), p)
        # vai giu
```

```

        elif i < n-1 and lab[i+1][j] == 0:
            push((i+1,j), p)
            # vai a destra
        elif j < n - 1 and lab[i][j+1] == 0:
            push((i,j+1), p)
            # vai a sinistra
        elif j > 0 and lab[i][j-1] == 0:
            push((i,j-1), p)
        # tutte le strade sono bloccate
        else:
            pop(p)
    return False

print(" --- LABIRINTO --- ")
lab1 = [
    [ 0, 0, 0, 1, 0 ],
    [ 0, 1, 1, 1, 0 ],
    [ 0, 0, 0, 1, 1 ],
    [ 0, 1, 0, 0, 0 ],
    [ 0, 0, 1, 1, 0 ] ]
print(risolvi(lab1))
lab2 = [
    [ 0, 0, 0, 1, 0 ],
    [ 0, 1, 1, 1, 0 ],
    [ 0, 0, 0, 1, 1 ],
    [ 0, 1, 0, 1, 0 ],
    [ 0, 0, 1, 1, 0 ] ]
print(risolvi(lab2))

```