Strutture dati in Python

Università Roma Tre – Dipartimento di Matematica e Fisica
M. Liverani, Laboratorio di ottimizzazione combinatoria

La libreria «pythonds»

È una libreria che fornisce classi di oggetti e metodi per operare su alcune strutture dati astratte:

- Stack: struttura dati sequenziale per la rappresentazione di una *pila* (struttura dati di tipo LIFO: *last in first out*)
- Queue: struttura dati sequenziale per la rappresentazione di una *coda* (struttura dati di tipo FIFO: *first in first out*)
- Deque: struttura dati simile alla coda, ma con due punti di ingresso e due punti di uscita
- BinarySearchTree: struttura dati per la rappresentazione di alberi binari di ricerca
- BinHeap: struttura dati per la rappresentazione di *heap* binari
- Graph: struttura dati per la rappresentazione di grafi e alberi generici

Maggiori informazioni su:

- http://interactivepython.org/runestone/static/pythonds/index.html
- http://www.mat.uniroma3.it/users/liverani/doc/pythonGraphs.pdf

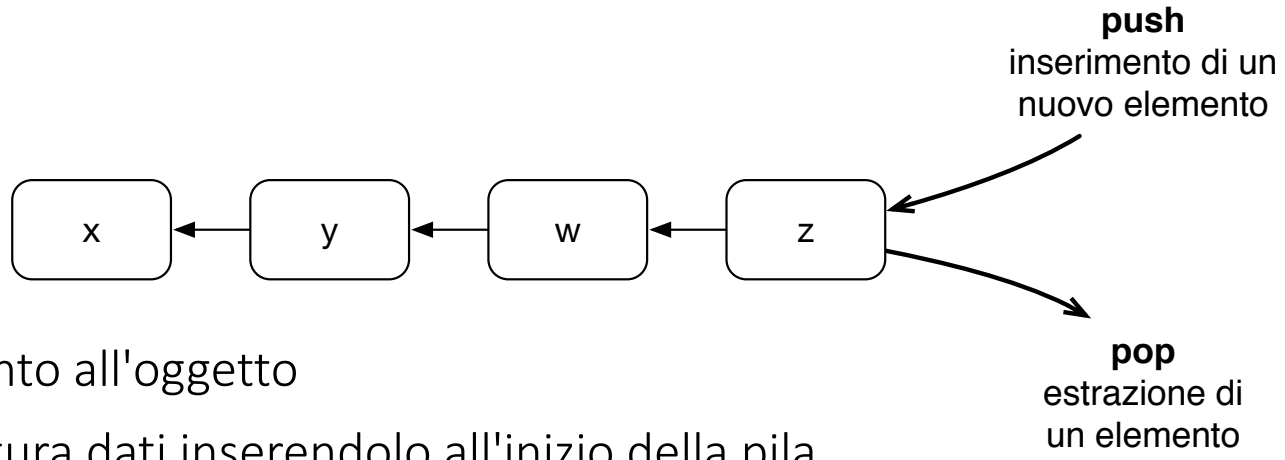
Installazione:

- `python3 -m pip install pythonds`

Utilizzo:

- `from pythonds import *`

La classe Stack

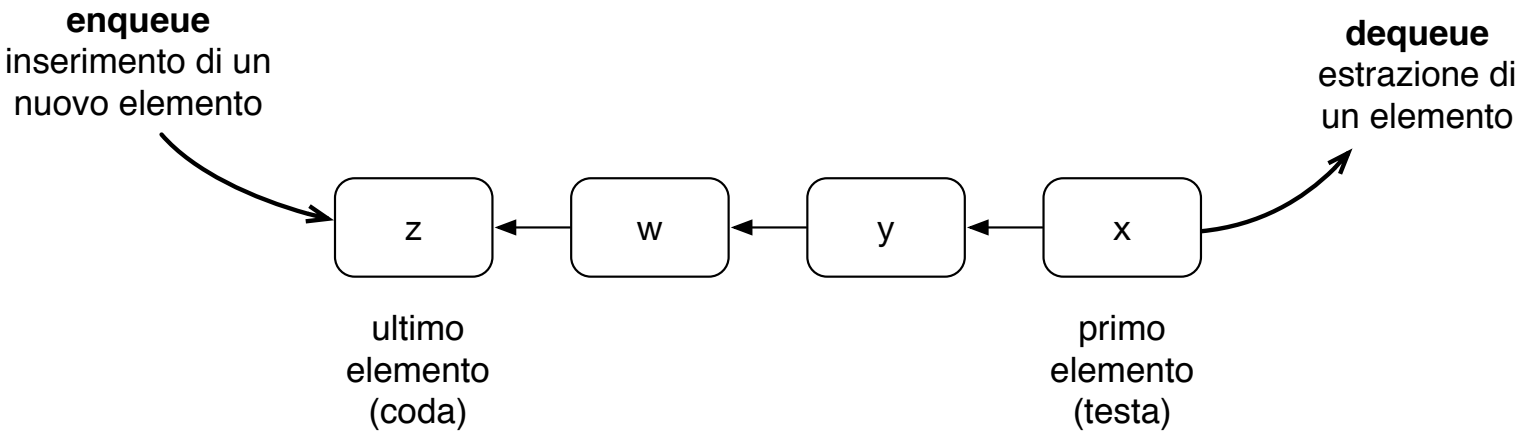


- Stack() : è il costruttore della classe;
  crea un nuovo *stack* vuoto e restituisce un riferimento all'oggetto
- push(*item*): aggiunge un nuovo elemento alla struttura dati inserendolo all'inizio della pila, come primo elemento
- pop() : restituisce l'elemento in cima alla pila e lo rimuove dallo *stack*
- peek(): restituisce il valore dell'elemento in cima alla pila, ma non lo rimuove dalla struttura dati. Lo *stack* non viene modificato
- isEmpty(): verifica se la pila è vuota e restituisce **True** se è vuota, **False** se contiene almeno un elemento
- size(): restituisce il numero di elementi presenti nella pila; restituisce un numero intero e non richiede alcun parametro
- Esempi:

```
a = Stack()
a.push(17)
b = a.pop()
```

```
if a.isEmpty() == True:
   print("La pila è vuota")
else:
   print("La pila contiene", a.size(), "elementi")
```

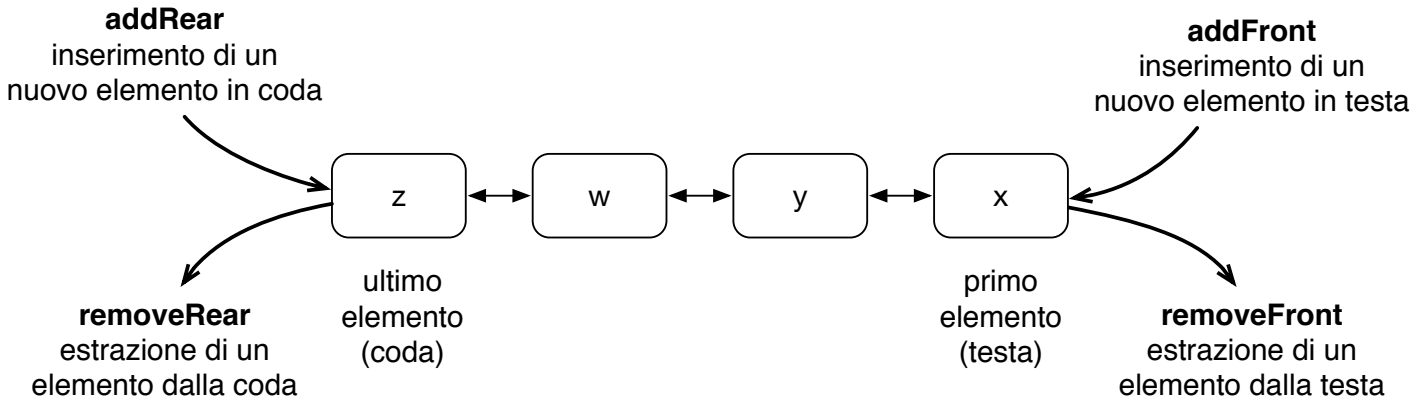
La classe Queue



- Queue(): è il metodo costruttore per istanziare gli oggetti della classe **Queue**; questo metodo crea una coda vuota e restituisce un riferimento alla struttura dati
- enqueue(*item*): accoda un nuovo elemento, aggiungendolo in fondo alla struttura dati come ultimo elemento. L'elemento da aggiungere alla coda deve essere passato come argomento
- dequeue(): restituisce l'elemento al primo posto nella coda e lo elimina dalla coda stessa
- isEmpty(): verifica se la coda è vuota, ossia priva di elementi; se la coda è vuota restituisce il valore booleano **True**, altrimenti, se la coda contiene almeno un elemento, restituisce **False**
- size(): restituisce il numero di elementi presenti nella coda; restituisce un numero intero e non richiede alcun parametro
- Esempi:

```
a = Queue()
a.enqueue("Pippo")
b = a.dequeue()
```

La classe Deque



- Deque(): è il metodo costruttore della classe; crea un nuovo oggetto della classe **Deque** vuoto e restituisce un riferimento all'oggetto creato
- addFront(*item*): aggiunge l'elemento specificato come argomento all'inizio della struttura dati
- addRear(*item*): aggiunge l'elemento specificato come argomento alla fine della struttura dati, come ultimo elemento della lista
- removeFront(): restituisce il primo elemento della lista e lo elimina dalla struttura dati
- removeRear(): restituisce l'ultimo elemento della lista e lo elimina dalla struttura dati
- isEmpty(): verifica se la struttura dati è vuota e restituisce un valore booleano: restituisce **True** se la lista è vuota, altrimenti restituisce **False**
- size(): restituisce il numero di elementi presenti nella struttura dati; restituisce un numero intero e non richiede alcun parametro

La classe BinarySearchTree

- BinarySearchTree(): è il metodo costruttore della classe; crea un nuovo oggetto vuoto della classe **BinarySearchTree** e restituisce un riferimento all'oggetto stesso
- put(*chiave*, *valore*): aggiunge un elemento alla struttura dati collocandolo nella posizione corretta nell'ambito dell'albero binario di ricerca; ciascun elemento è composto da una coppia: la ***chiave*** è l'attributo che determina la posizione nell'albero binario di ricerca, mentre il ***valore*** è l'informazione che si intende memorizzare nella struttura dati
- get(*chiave*): restituisce il valore del primo elemento identificato con la chiave specificata come argomento presente nell'albero. L'elemento non viene rimosso dalla struttura dati. Se non esiste un elemento con tale chiave, restituisce **None**
- delete(*chiave*): elimina dalla struttura dati il primo elemento presente nell'albero identificato dalla chiave specificata come argomento. Se non esiste alcun elemento identificato dalla chiave specificata, produce un errore e blocca il programma
- length(): restituisce il numero di elementi presenti nell'albero binario di ricerca; se l'albero è vuoto, restituisce 0.
- inorder(): esegue una visita in profondità dell'albero binario a partire dalla radice; per ogni vertice *v* dell'albero, visualizza le chiavi dei nodi visitati nel seguente ordine:
  - visualizza le chiavi del sotto-albero con radice in *left(v)*
  - visualizza la chiave di *v*
  - visualizza le chiavi del sotto-albero con radice in *right*(*v*)

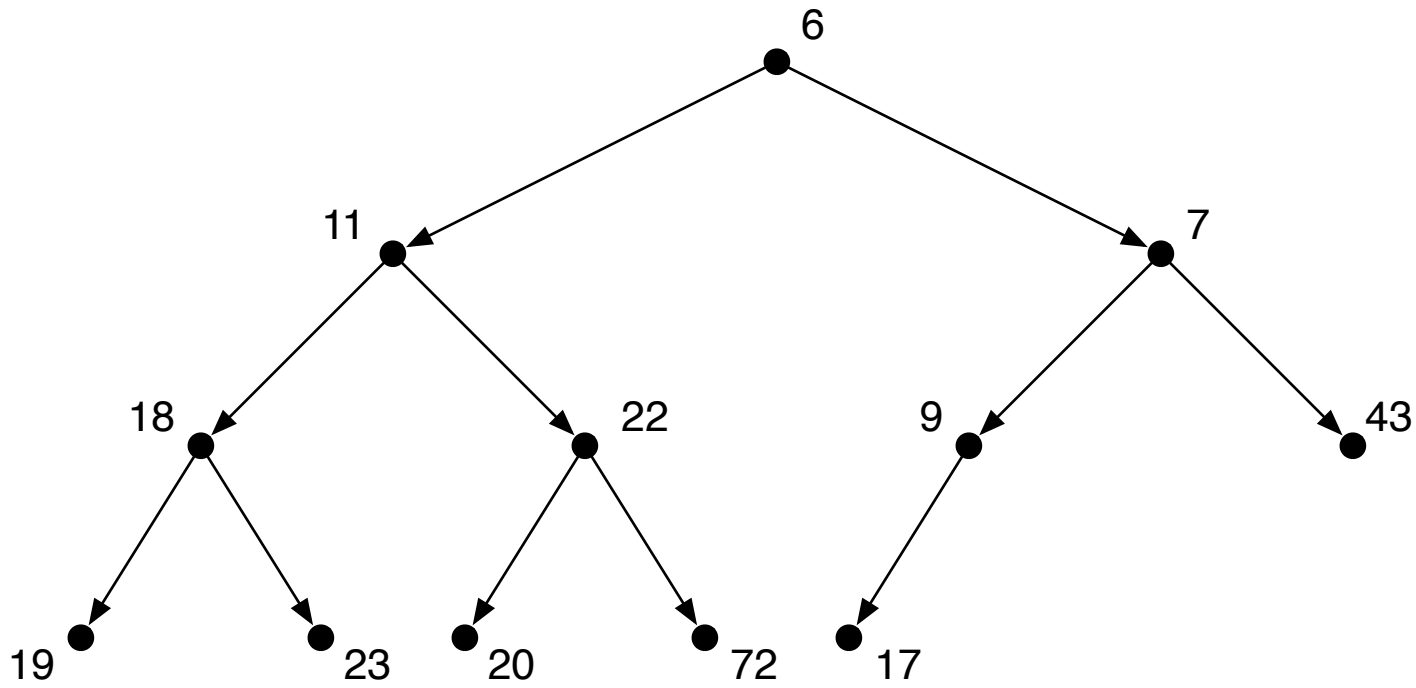
  In questo modo le chiavi vengono visualizzate in ordine crescente.
- postorder(): esegue una visita dell'albero binario in «post-ordine» a partire dalla radice; per ogni vertice *v* dell'albero, visualizza le chiavi dei nodi visitati nel seguente ordine:
  - visualizza le chiavi del sotto-albero con radice in *right*(*v*)
  - visualizza le chiavi del sotto-albero con radice in *left*(*v*)
  - visualizza la chiave di *v*
- in: l'operatore restituisce **True** se una determinata chiave è presente nell'albero, altrimenti restituisce **False**

La classe BinHeap

- BinHeap(): è il metodo costruttore della classe; crea un nuovo oggetto vuoto della classe **BinHeap** e restituisce un riferimento all'oggetto stesso
- insert(*k*): inserisce l'elemento *k* nell'heap
- delMin(): restituisce l'elemento minimo presente nell'heap e lo elimina dalla struttura dati

- Esempi:

```
a = BinHeap()
a.insert(17)
b = a.delMin()
```



La classe Graph

- Graph(): è il metodo costruttore della classe, crea un oggetto **Graph** e restituisce un riferimento all'oggetto stesso
- addVertex*(v)*: aggiunge il vertice v al grafo. Il vertice è un oggetto della classe **Vertex**
- addEdge(*u, v*): aggiunge lo spigolo (u,v) al grafo; lo spigolo è orientato: uscente dal vertice u ed entrante nel vertice v
- addEdge(*u, v, w*): aggiunge lo spigolo pesato (u,v) al grafo; lo spigolo è orientato da u a v ed ha un «peso» w
- getId(): se il vertice v appartiene al grafo, restituisce l'identificativo del vertice
- getVertex(*x*): se presente, restituisce l'oggetto **Vertex** con identificativo x
- getVertices(): restituisce la lista degli identificativi dei vertici del grafo
- in: restituisce il valore booleano **True** se il vertice appartiene al grafo, altrimenti restituisce **False**
- Esempi:

```
g = Graph()
g.addVertex(1)
g.addVertex(2)
g.addEdge(1,2)
```

```
v = g.getVertex(1)
V = getVertices()
print("Vertici:", V)
```

La classe Vertex

- getConnections(): restituisce la lista di adiacenza del vertice (la lista dei vertici adiacenti)
- setColor(*colore*): assegna il colore specificato al vertice; il colore può essere indifferentemente un numero o una stringa di caratteri e rappresenta semplicemente un attributo informativo che caratterizza il vertice
- getColor(): restituisce il «colore» (numero intero o stringa) assegnato al vertice del grafo con il metodo **setColor()**
- setDistance(*d*): assegna la distanza d al vertice; anche in questo caso si tratta di un attributo informativo che caratterizza il vertice, il cui significato dipende dal contesto in cui lo si utilizza
- getDistance(): restituisce il valore di una distanza assegnato al vertice con il metodo **setDistance()**
- Esempi:

```
v = g.getVertex(3)
v.getId() → restituisce 3
v.setColor(1)
c = v.getColor()
v.setDistance(u.getDistance() + 1)
v.setPred(u)
padre = v.getPred()
```

Creazione di un grafo completo

```
from pythonds import *

def completeGraph(G, n):
    for v in range(1,n+1):
        G.addVertex(v)
    for u in range(1,n):
        for v in range(u+1,n+1):
            G.addEdge(u,v)
            G.addEdge(v,u)
    return
```

```
g = Graph()
n = int(input("Numero di vertici:  "))
completeGraph(g, n)
print("Vertici del grafo: ", g.getVertices())
print("Spigoli del grafo:")
for u in g:
    for v in u.getConnections():
        print("(%s,%s)" % (u.getId(), v.getId()))
```