Bags, Queueues e STacks

Queste strutture dati sono fondamentali e largamente diffuse ed utili. L'obbiettivo di questa sezione è quello di dimostrare l'importanza delle strutture dati linkate. In particolare, la struttura linked list permette l'implementazione di bags, queues e stacks.

APIs

Ogni struttura contiene un costruttore privo di argomenti, un metodo per aggiungere items alla collezione, un metodo per testare se la collezione è vuota ed un metodo che ritorna la grandezza della collezione.

```
public class Bag<Item>
        Bag()              Crea una bag vuota
    void add(Item item)    aggiunge un item
 boolean isEmpty           check
     int size              numero di elementi

Public class Queue<Item>
        Queue()                    crea una queue vuota
   voiol enqueue(Item item)        add item
    Item dequeue()                 rimuove l'ultimo el aggiunto
 boolean isEmpty()                 check
     int size()                    numero di elementi

Public class Stack()<Item>
        Stack()    Crea Stack vuoto
    void push() add    Item pop() rimuove last el
 bodean isEmpty()      ckeck
     int size()        numero di el
```

Bags

Una bag è una collezione dove la rimozione di un elemento NON è supportata. Inoltre, l'ordine di iterazione non è specificato, e dovrebbe non avere importanza per l'utente.
Un esempio di applicazione potrebbe essere il seguente: consideriamo l'immissione nello stdinput una serie di numeri, con l'obiettivo di farne la media.
Per questo tipo di calcolo non ci interessa l'ordine con cui vengono prelevati gli elementi.

FIFO QUEUES

Una queue è una struttura basata sul principio FIFO. La politica di effettuare dei task nell'ordine in cui essi arrivano è collegabile al concetto 'reale' di 'EQUITÀ'. Potremmo usare questa struttura, ad esempio, per popolare un array di elementi letti da un file, nell'ordine in cui sono letti:

```
for (int i = 0; i < N; i++)
        array[i] = queue.dequeue();
```

Pushdown Stack

Uno stack è una struttura che utilizza la politica 'LIFO': quando impiliamo dei documenti, stiamo utilizzando uno stack. Ci sono molte applicazioni di uno stack: ad esempio, la cronologia di un browser è basata su uno stack; effettuiamo un push() quando apriamo una nuova pagina ed un pop() quando torniamo indietro.

RESIZE DI ARRAY

Scegliere un array per rappresentare uno STACK implica che l'utente debba stimare la grandezza massima dello stack in precedenza. Un utente che, però, sceglie una grande capacità rischia di sprecare dello spazio nel momento in cui l'array è per la maggior parte vuoto. Allo stesso modo si rischia di avere un overflow nel momento in cui si fa un push() in un array pieno.

Per questo motivo, la push() ha bisogno di una parte di codice che TESTI se abbiamo uno STACK pieno.
Quindi, dopo aver controllato se l'array è pieno (nella push()), abbiamo 2 possibilità:

1) L'array è pieno:

```
if (N == a.length) resize (2 x a.length);
```

ovvero RADDOPPIAMO la capacità dell'array

2) L'array ha spazio:

```
a[N++] = item;
```

semplicemente aggiungiamo l'el.

Riduzione dell'array

In un modo simile, riduciamo la grandezza dell'array nella funzione pop() quando ci accorgiamo che la grandezza complessiva è più grande degli elementi presenti all'interno.
Potremmo pensare di ridurre, in modo analogo, quando gli elementi sono ((arr.size())/2)-1, ma questo è un errore.
Ciò che invece è corretto fare, è ridurre l'array per 1/2 quando esso è pieno per l'1/4 della capienza.

```
IF (N > 0 && N == a.length/4) resize(a.length/2)
```

In questo modo l'array non avrà mai un overflow né sarà di capienza pari ad 1/4..

Loitering

Il garbage collector di Java ha come obbiettivo quello di 'pulire' automaticamente dalla memoria gli elementi che non possono più essere acceduti.
Il problema, è che usando gli array per implementare gli STACK, nel momento in cui eseguiamo un pop() di un elemento, il riferimento all'elemento persiste nell'array.
Di conseguenza il Garbage Collector non ha idea che quell'elemento non è più usato.

La soluzione a questo problema non è troppo difficile: basta includere nel codice della pop() una porzione di codice che imposti a null il riferimento all'elemento eliminato.