Algoritmi e Strutture Dati

Strutture dati elementari:

Esercitazione

Domenico Fabio Savo

Il tipo di dato Pila

tipo `Pila`:
dati:
una sequenza S di n elementi.

operazioni:

`isEmpty`$() \rightarrow result$
restituisce `true` se S è vuota, e `false` altrimenti.

`push`$(elem\ e)$
aggiunge e come ultimo elemento di S.

`pop`$() \rightarrow elem$
toglie da S l'ultimo elemento e lo restituisce.

`top`$() \rightarrow elem$
restituisce l'ultimo elemento di S (senza toglierlo da S).

Pila: implementazione tramite array

La pila sarà rappresentata come una coppia (**S**, **t**) dove:

- **S** è un array di dimensione N
- **t** è un intero che rappresenta l'indice dell'elemento affiorante della pila.

Pila: esercizio

Esercizio:

Scrivere lo pseudocodice che implementa le operazioni definite sul tipo di dato PILA (*isEmpty*, *push*, *pop*, *top*), nel caso in cui questo è implementato tramite un array.

Pila: isEmpty

- Il primo indice dell'array è zero, quindi inizializziamo **t** a -1.

isEmpty() → *true* or *false*

if (t < 0) **then return** *true*

else return *false*

Pila: push

```
push(elem e)
        if (t = N-1)
        then lancia eccezione la pila è piena;
        else { t ← t+1;
                S[t] ← e; }
```

Nel caso in cui l'array è pieno, invece di lanciare l'eccezione si potrebbe adottare una politica di ridimensionamento dell'array utilizzato. Come si può fare?

Pila: pop

```
pop() → elem e
        if (S.isEmpty())
        then return null;
        else { elem e ← S[t];
                S[t] ← null;
                t ← t-1;
                return e; }
```

Pila: top

```
top() → elem e
        if (S.isEmpty())
        then return null;
        else return S[t];
```

Pila: esercizio

Esercizio 2:

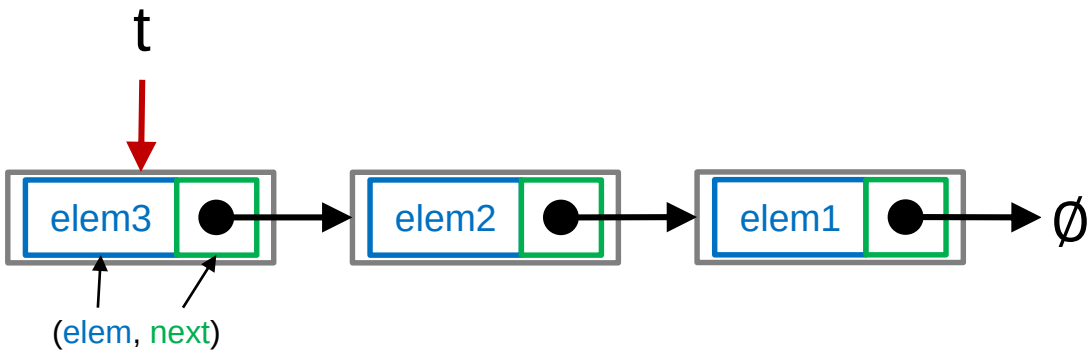
Scrivere lo pseudocodice che implementa le operazioni definite sul tipo di dato PILA (*isEmpty*, *push*, *pop*, *top*), nel caso in cui questo è implementato tramite una **lista semplice**.

Pila: implementazione mediante lista semplice

La pila sarà rappresentata come una coppia (**L**, **t**) dove:

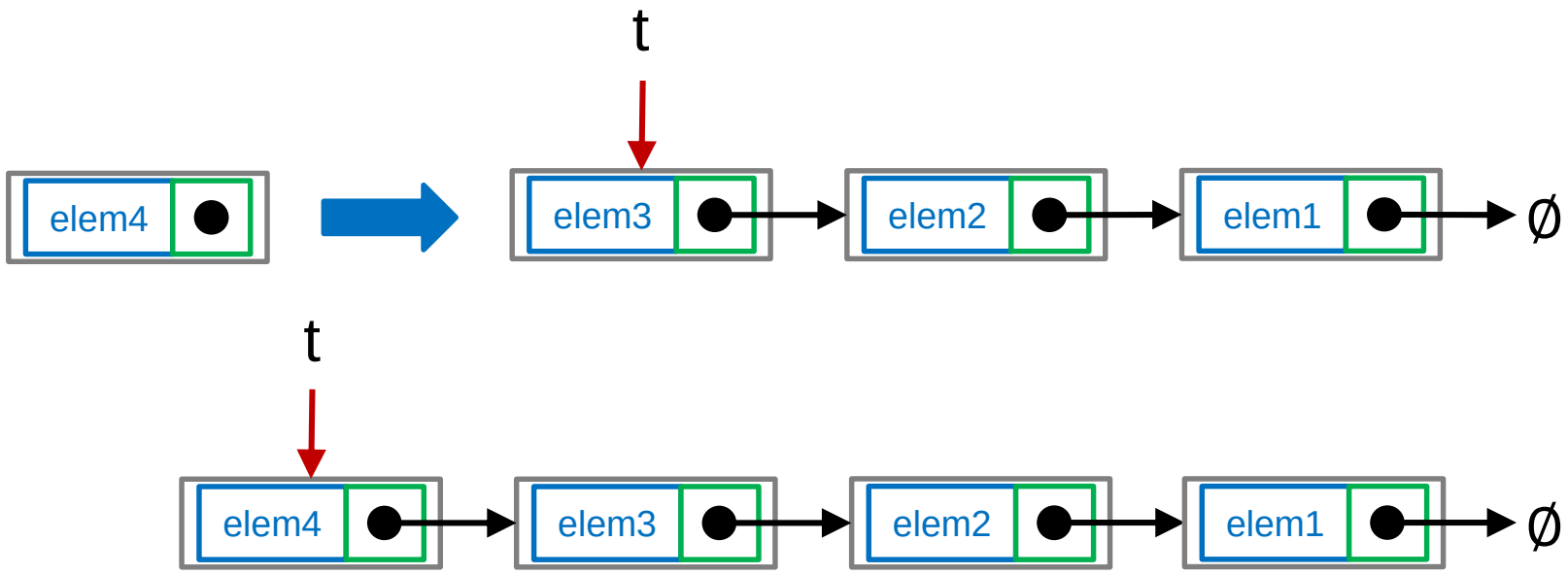
- **L** è una lista semplice in cui ogni record è una coppia (*elem*, *next*), dove *next* è il puntatore al successivo record della lista
- **t** è il puntatore all'elemento affiorante della pila

Pila: implementazione mediante lista semplice



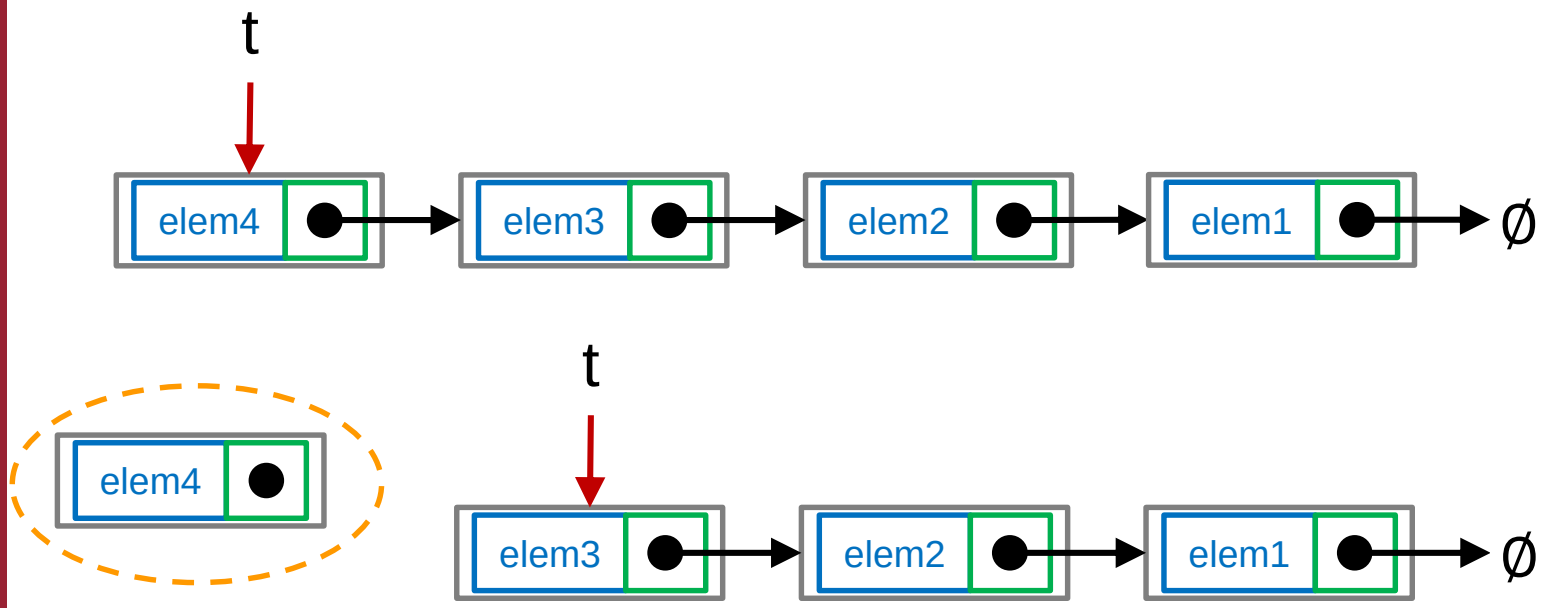
- Il record **t** rappresenta sempre l'elemento affiorante della pila.
- L'ultimo record della pila punterà (valore *next*) al record nullo (*null*).

Pila: inserimento record



- L'aggiunta di un nuovo record nella pila è realizzata attraverso l'aggiunta di un nuovo record in testa alla lista.

Pila: estrazione record



- L'estrazione di un record dalla pila è realizzata attraverso l'eliminazione del **primo** elemento della lista (ovvero il record **t**).

Pila: isEmpty

- Se la pila non contiene alcun elemento allora **t** punterà all'elemento nullo.

isEmpty() → *true* or *false*

if (t = *null*) **then return** *true*

else return *false*

Pila: push

push(elem e)

sia r un nuovo record;

r.elem ← e;

r.next ←t;

t ←r;

- Inseriamo un nuovo record all'inizio della lista.

Pila: pop

```
pop() → elem e
        if (L.isEmpty()) then return null;
        else { r ← t;
               t ← r.next;
               return r.elem; }
```

- Il record restituito sarà il primo record della lista.

Pila: top

top() → elem e

if (L.isEmpty())

then return *null*;

else return t.elem;

Il tipo di dato Coda

tipo `Coda`:
dati:
una sequenza S di n elementi.

operazioni:

`isEmpty`$() \rightarrow result$
restituisce `true` se S è vuota, e `false` altrimenti.

`enqueue`$(elem\ e)$
aggiunge e come ultimo elemento di S.

`dequeue`$() \rightarrow elem$
toglie da S il primo elemento e lo restituisce.

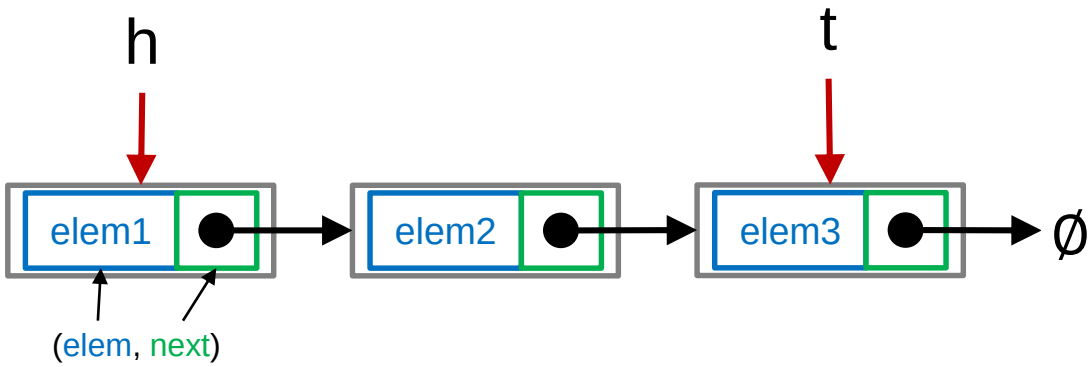
`first`$() \rightarrow elem$
restituisce il primo elemento di S (senza toglierlo da S).

Coda: implementazione mediante lista semplice

La coda sarà rappresentata come una tripla (**L**, **h, t**) dove:

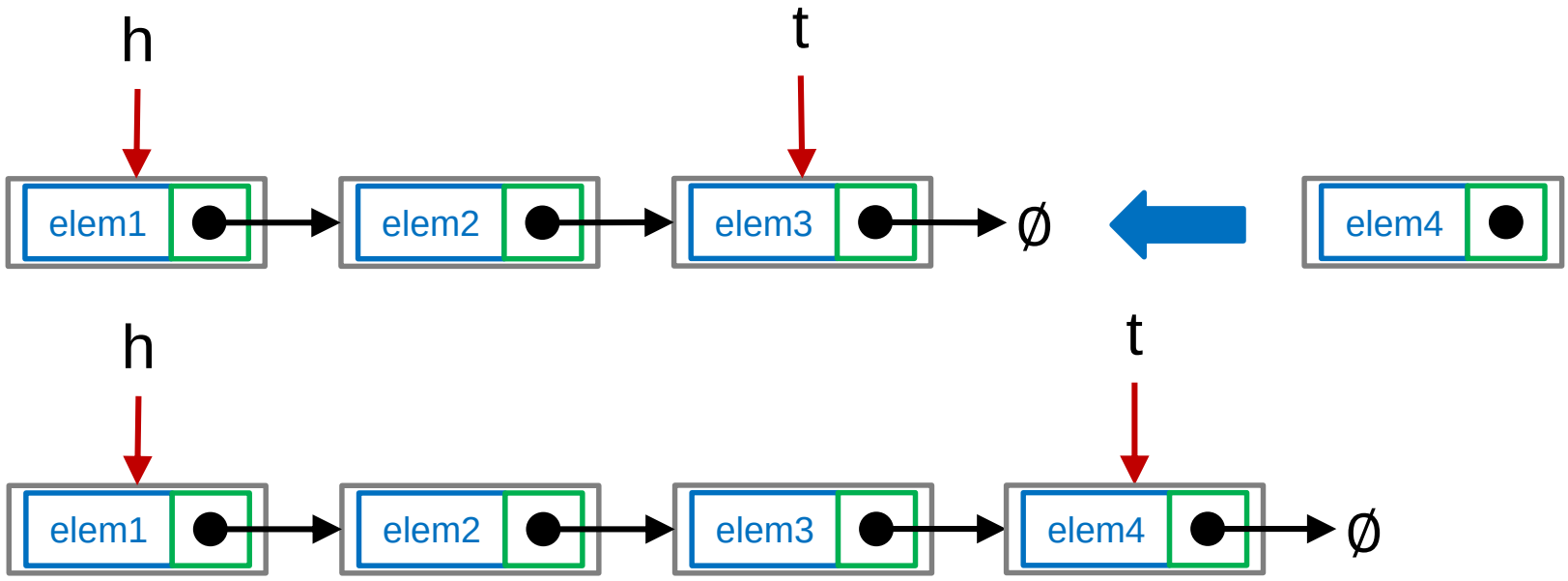
- **L** è una lista semplice in cui ogni record è una coppia (*elem*, *next*), dove *next* è il puntatore al successivo record della lista
- **h** è il puntatore al primo record della lista
- **t** è il puntatore all'ultimo record della lista

Coda: implementazione mediante lista semplice



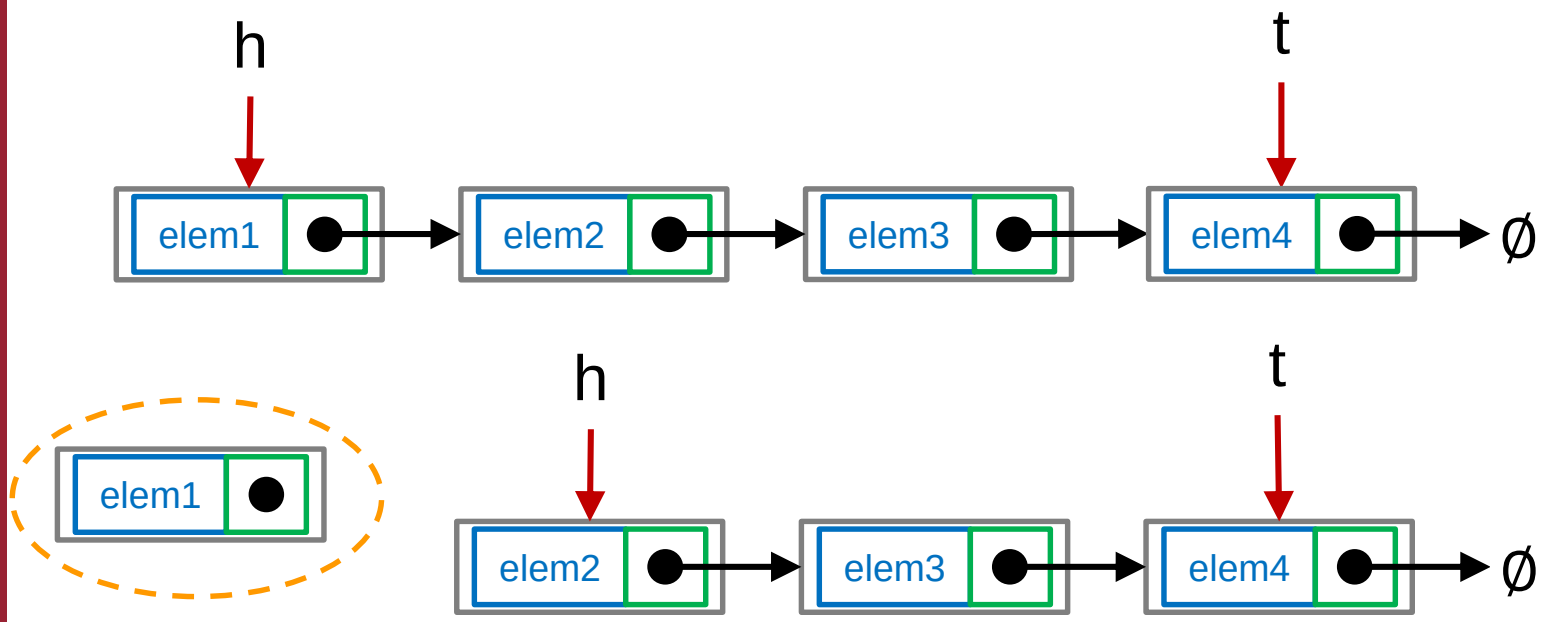
- Il record **h** è il primo elemento della lista e rappresenta la testa della coda. Il record **t** è l'ultimo elemento della lista e rappresenta la fine della coda.
- L'ultimo record della lista (ovvero il record **t**) punterà (valore *next*) all record nullo (*null*).

Coda: inserimento record



- L'aggiunta di un nuovo record nella coda è realizzata attraverso l'aggiunta di un nuovo record alla fine della lista.

Coda: estrazione record



- L'estrazione di un record dalla coda è realizzata attraverso l'eliminazione del **primo** elemento della lista (ovvero il record **h**).
- **DOMANDA:** e se volessimo eliminare non il **primo** elemento della lista, ma l'**ultimo**?

Coda: esercizio

Esercizio:

Scrivere lo pseudocodice che implementa le operazioni definite sul tipo di dato CODA (*isEmpty*, *enqueue*, *dequeue*, *first*), nel caso in cui questo è implementato tramite una lista semplice.

Coda: isEmpty

- Se la coda non contiene alcun elemento allora **h** sarà uguale a **t** ed entrambi saranno nulli.

isEmpty() → *true* or *false*

if (t = *null*) **then return** *true*

else return *false*

Coda: enqueue

```
enqueue(elem e)
        r.elem ← e;
        if(L.isEmpty()) then {
                h ←r; t ←r; r.next ←null; }
        else { t.next ←r;
                r.next ←null;
                t ←r; }
```

- Inseriamo un nuovo record alla fine della lista.

Coda: dequeue

```
dequeue() → elem e
        if (L.isEmpty()) then return null;
        else { r ← h;
               h ← r.next;
               return r.elem; }
```

- Il record restituito sarà il primo record della lista.

Coda: first

```
first() → elem e
        if (L.isEmpty())
        then return null;
        else return h.elem;
```

Bilanciamento delle parentesi

Diciamo che le parentesi in una stringa di caratteri sono **bilanciate** se per ogni parentesi aperta presente nella stringa è presente anche la sua parentesi chiusa.

- **xy{zkzk{yxy}htht{f}kzkz}yx → parentesi bilanciate**
- **xy{vv{kkk}vvv}yyy}xyxy{xx → parentesi non bilanciate**

Bilanciamento delle parentesi

Esercizio: scrivere un algoritmo il quale, data una stringa S, verifichi che le parentesi contenute in S siano bilanciate.

Suggerimento: utilizzare una **PILA**.

Bilanciamento delle parentesi

Soluzione

```
c ← «primo carattere di S»;
while (not «fine di S») {
        if (c = '{') then p.push('1');
        if (c = '}')
                if (p.isEmpty()) then
                        return  false;
                else p.pop();
        c ← «prossimo carattere di S»;
}
if ( not p.isEmpy()) then return false;
return true;
```

S = y{ztzt{xyxy{tt}yxyx}ztzt}