ALGORITMI E STRUTTURE DI DATI

Insieme dinamici

Sono insieme da cui si possono aggiungere e togliere elementi, in particolare di oggetti. I campi che costituiscono l'elemento sono caratterizzati da una chiave, che identifica l'oggetto e consente un ordinamento, e i dati satellite, valori che servono all'applicazione.

Lo pseudocodice è privo di definizioni di tipo. Le variabili sono allocate in memoria nel momento in cui vengono menzionate per la prima volta.

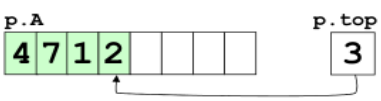
I sistemi dinamici si distinguono anche per il tipo di operazioni supportate (operazioni di consultazione e di modifica).

Pila e Code

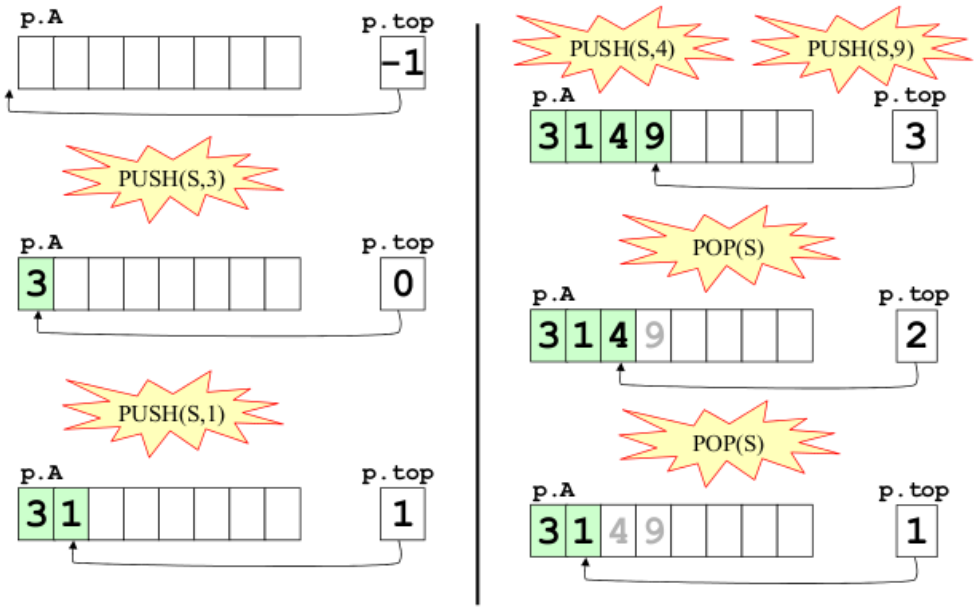
La pila usa una strategia LIFO, mentre la coda usa una strategia FIFO.

Pila

Nella pila l'operazione di consultazione riguarda la **IS-EMPTY** (dove torna "true" se la pila è vuota, altrimenti torna "false"), mentre tra le operazioni di modifica si trova il **PUSH** (l'inserimento di un elemento nella pila) e **POP** (rimozione e restituzione di un elemento affiorante nella pila). Altre operazioni possono essere la **TOP** (ritorna l'elemento affiorante senza rimuoverlo, quando è vuota vale -1), l'**EMPTY** (svuota la pila) e **SIZE** (ritorna il numero di elementi in pila). Il tempo di esecuzione di ogni operazione è $\Theta(1)$.



sequenza di operazioni su una pila




```
PUSH(p,x)
1.  if p.top == p.A.length-1
2.      then   error "overflow"
3.      else   p.top = p.top + 1
4.             p.A[p.top] = x
```

```
POP(p)
1.  if p.top == -1
2.      then   error "underflow"
3.      else   p.top = p.top - 1
4.             return p.A[p.top + 1]
```

```
IS-EMPTY(p)
1.  return p.top == -1          ▷ true se la pila è vuota
```

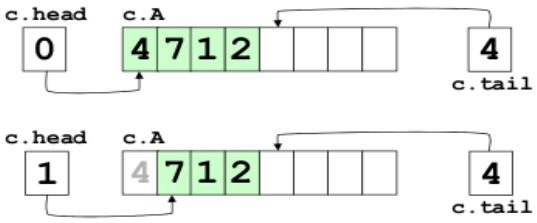
```
EMPTY(p)
1.  p.top = -1                  ▷ vuoto la pila
```

```
TOP(p)
1.  return p.A[p.top]           ▷ l'elemento affiorante
```

```
SIZE(p)
1.  return p.top + 1                ▷ il numero di elementi
```

Code

Con le code possiamo avere come operazioni di consultazione la **IS-EMPTY** (ritorna "true" se la coda è vuota, altrimenti "false"), per le operazioni di modifica **ENQUEUE** (inserisce un elemento nella coda) e **DEQUEUE** (rimuove l'elemento più vecchio nella coda). Altre operazioni possono essere il **FRONT** (ritorna l'elemento più vecchio senza rimuoverlo), l'**EMPTY** (svuota la coda) e il **SIZE** (ritorna il numero di elementi in coda). In una coda avremo due elementi fondamentali: l'**HEAD** (l'elemento più vecchio) e il **TAIL** (l'elemento in prima posizione utile). Quando entrambi puntano la stessa casella la coda è vuota.
Il tempo di esecuzione di ogni operazione è $\Theta(1)$.




```
ENQUEUE(c,x)
1.  if c.head==c.tail+1 or (c.tail==c.A.length-1 and c.head==0)
2.      then    error "overflow"
3.      else    c.A[c.tail] = x
4.              if c.tail == c.A.length-1
5.                  then    c.tail = 0
6.                  else    c.tail = c.tail + 1
```

```
DEQUEUE(c)
1.  if c.head == c.tail
2.      then    error "underflow"
3.      else    x = c.A[c.head]
4.              if c.head == c.A.length-1
5.                  then    c.head = 0
6.                  else    c.head = c.head + 1
7.              return x
```

```
IS-EMPTY(c)
1.  return c.head == c.tail
```

```
EMPTY(c)
1.  c.head = c.tail = 0
```

```
FRONT(c)
1.  if c.head == c.tail
2.      then    error "empty queue"
3.      else    return c.A[c.head]
```

Liste

Gli oggetti sono disposti in ordine lineare, determinato dai dati ed un eventuale chiave, dalla successione degli elementi oppure senza nessuna regola precisa; prevedono tutte le operazioni dei sistemi dinamici. Tra le operazioni di aggiornamento troviamo anche l'**INSERT(l,x)** (inserisce l'elemento x in testa alla lista l), l'**INSERT-BEFORE(l,x,y)** (inserisce l'elemento x prima dell'elemento y), l'**ADD(l,x)** (aggiungere l'elemento x in coda alla lista l), l'**ADD-AFTER(l,x,y)** (aggiunge l'elemento x dopo l'elemento y), il **DELETE(l,x)** (rimuove l'elemento x dalla lista l) e l'**EMPTY(l)** (vuota la lista). Tra le operazioni di consultazione invece troviamo il **NEXT(l,x)** (ritorna l'elemento che segue x oppure NIL se x è l'ultimo elemento), **PREV(l,x)** (ritorna l'elemento che precede x nella lista oppure NIL se x è il primo elemento), **FIRST(l)** (ritorna il primo elemento della lista), **LAST(l)** (ritorna l'ultimo elemento), **SEARCH(l,k)** (cerca l'elemento chiave nella k nella lista l) e **IS-EMPTY(l)** (ritorna "true" se la lista è vuota, altrimenti torna "false").

Le liste si possono implementare in vari modi:

Lista Concatenata

```
IS-EMPTY(l)
1.  return l.head == NIL
```

```
EMPTY(l)
1.  l.head = NIL ▷ lo pseudocodice non dealloca memoria
```

```
FIRST(l)
1.  ret
```

```
NEXT(l,x)
```

```
INSERT(l,x)
```

```
DELETE-FIRST(l)
1.  ▷ NOTA: lo pseudocodice non dealloca l'elemento
2.  if l.head == NIL
3.      then    error "lista vuota"
4.      else    l.head = l.head.next
```

Ha come attributi ".next", ".key" e ".info".

Lista doppiamente concatenata

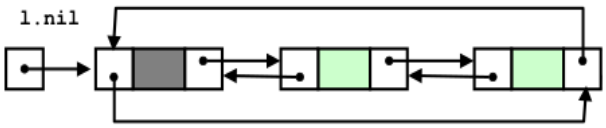
Ha come attributo anche ".prev" e ".tail".

```
INSERT(l,x)
1. x.next = l.head
2. if l.head ≠ NIL
3.     then   l.head.prev = x
4. l.head = x
5. x.prev = NIL
```

```
DELETE(l,x)
1. if x.prev ≠ NIL
2.     then   x.prev.next = x.next
3.     else   l.head = x.next
4. if x.next ≠ NIL
5.     then   x.next.prev = x.prev
```

Sentinella

È un nodo fittizio introdotto in testa alla lista. Diventa una lista circolare; una lista vuota contiene solo la sentinella.




```
DELETE(l,x)
1. x.prev.next = x.next
2. x.next.prev = x.prev
```

```
LIST-INSERT(l,x)
1. x.next = l.nil.next
2. l.nil.next.prev = x
3. l.nil.next = x
4. x.prev = l.nil
```

```
LIST-SEARCH(l,k)
1. x = l.nil.next
2. while x ≠ l.nil and x.key ≠ k
3.     do x = x.next
4. return x
```