PILE E CODE

Strutture di dati utilizzate per la gestione di sequenze lineari dinamiche

Pila: collezione di elementi in cui le operazioni disponibili, come l'estrazione di un elemento, sono ristrette unicamente a quello più recentemente inserito

Politica di accesso detta LIFO (Last In First Out)

Operazioni disponibili

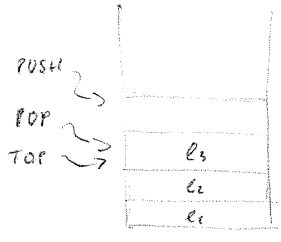
PUSH: inserisce un nuovo elemento in cima alla pila

POP: estrae l'elemento in cima alla pila

TOP: restituisce (legge, senza estrarre) l'elemento in cima alla pila.

EMPTY: verifica se la pila è vuota (Utile in alcune applicazioni)

Ogni operazione invocata su una pila può essere eseguita in tempo costante indipendentemente dalla dimensione della pila (per analogia, si pensi ad una pila di vassoi).

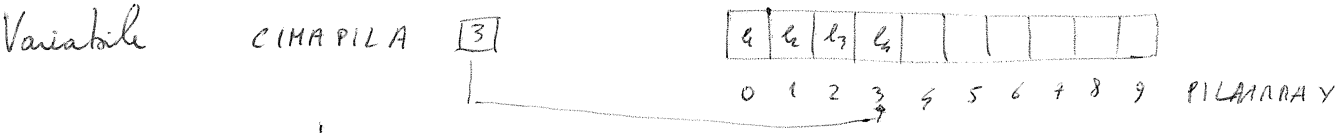


Implementazione

Una pila può essere implementata usando gli array dinamici: la dimensione iniziale è costante, poi viene raddoppiata e dimezzata a seconda delle operazioni effettuate.

Gli elementi della pila sono memorizzati in sequenza nell'array, a partire dalla locazione iniziale, inserendoli man mano nella prima locazione disponibile.

La cima corrisponde all'ultimo elemento della sequenza.
Occorre tener traccia dell'indice della locazione che contiene l'ultimo elemento per implementare PUSH, POP, TOP e EMPTY.



Inizialmente uguale a -1.

TOP restituisce PilaArray[CIMAPILA]

PUSH incrementa CIMAPILA (dopo aver verificato che l'array non è pieno) e inserisce l'elemento in PilaArray[CIMAPILA]

POP restituisce PilaArray[CIMAPILA] e decrementa CIMAPILA (dopo aver verificato che l'array non debba essere ridotto)

EMPTY restituisce l'esito del controllo CIMAPILA == -1 (pila vuota)

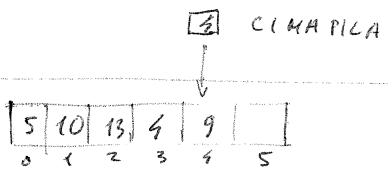
In codice ...

```
PUSH (x):
  VerificaRaddoppio ();
  CIMAPILA = CIMAPILA + 1;
  PilaArray [CIMAPILA] = x;

POP ():
  IF (!EMPTY()) {
     x = PilaArray [CIMAPILA];
     CIMAPILA = CIMAPILA - 1;
     VerificaDimezzamento ();
     RETURN x;
  }

TOP ():
  IF (!EMPTY()) RETURN PilaArray [CIMAPILA];

EMPTY ():
  RETURN (CIMAPILA == -1);
```



TOP () restituisce 9

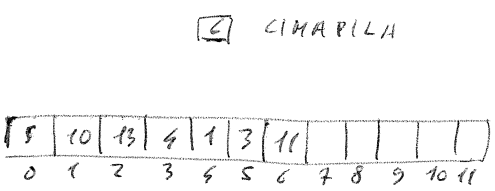
EMPTY () restituisce FALSE

POP () restituisce 9 e pone CIMAPILA a 3

PUSH (1) inserisce 1 e pone CIMAPILA a 4

PUSH (3) inserisce 3 e pone CIMAPILA a 5

PUSH (11) provoca il raddoppio, inserisce 11 e pone CIMAPILA a 6



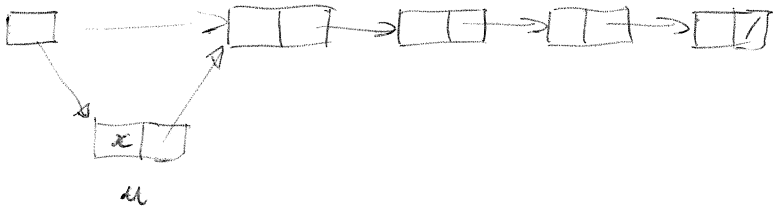
Una pila può essere implementata anche usando una lista i cui elementi sono mantenuti ordinati in base al loro tempo di inserimento

La cima della pila corrisponde all'inizio della lista.
Le operazioni agiscono tutte sull'elemento iniziale.

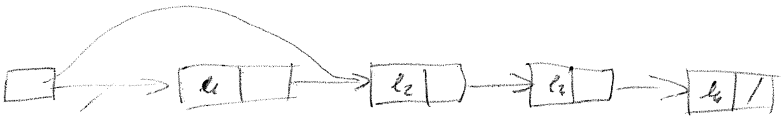
Variabile CIMAPILA (memorizza un puntatore al primo elemento)



```
PUSH (x):
  u = NuovoNodo();
  u.dato = x;
  u.succ = CIMAPILA;
  CIMAPILA = u;
```




```
POP ();
IF (!EMPTY()){
  x = CIMAPILA.dato;
  CIMAPILA = CIMAPILA.succ;
  RETURN x;
}
```



$x = e_1$

IMPLEMENTAZIONE
È corretta secondo voi ??

```
TOP:
  IF (!EMPTY()) RETURN CIMAPILA.dato;

EMPTY:
  RETURN (CIMAPILA == NULL)
```

Le modalità di allocazione di un nodo dipendono dal linguaggio di programmazione.

OPUS LIBRI

OPUS DEI

Come il divino permette
di progredire spiritualmente
attraverso la sua azione...

OPUS LIBRI

... così un libro permette di
progredire mentalmente attraverso
l'applicazione dei suoi contenuti.

Non avremo il tempo di veder tutti gli esempi
a lezione ma ... studiateli tutti!

Postscipt e notazione postfissa

Postscipt: linguaggio di programmazione per la grafica, eseguito da un interprete che usa una PILA e la notazione postfissa (anche detta POLACCA INVERSA)

Postscipt, standard tipografico, evoluto in PDF (Portable Document Format)

Notazione postfissa = operatori "alla fine"

Notazione abituale in matematica: A + B (Notazione infissa)

operatore

operando operando

Notazione postfissa : A B +

operandi operatore

Introdotta e studiata dal matematico Lukasiewicz.

Vantaggi:

- non c'è bisogno di parentesi, l'ordine di valutazione degli operatori non è ambiguo
- non è necessario specificare priorità per gli operatori

  $A + B \times C \equiv A + (B \times C)$ è dovuto alla convenzione che $\times$ ha priorità su $+$.
- tali espressioni si prestano ad essere valutate semplicemente da sinistra a destra mediante l'uso di una PILA.

Regole di valutazione

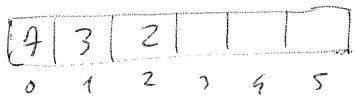
- operando : viene eseguita una PUSH dell'operando
- operatore : vengono eseguite due POP, l'operatore viene applicato ai due operandi prelevati dalla pila e viene eseguita una PUSH del risultato.

Esempio 732+X

Scandiamo l'espressione da sinistra a destra.

Push(7), Push(3), Push(2)

CIMA PILA [2]



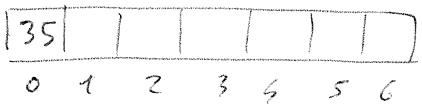
In corrispondenza di +, Pop(), Pop() restituiscono 2 e 3, 2+3=5, Push(5) produciamo

CIMA PILA [1]



In corrispondenza di X, Pop(), Pop() restituiscono 5 e 7, 5x7=35, Push(35) produciamo

CIMA PILA [0]



Pertanto la pila è un'ottima struttura di dati di ausilio alla VALUTAZIONE di espressioni algebriche in notazione postfissa.

In realtà è molto utile anche per CONVERTIRE un'espressione algebrica in forma infissa in un'espressione algebrica equivalente in forma postfissa.

Supponiamo per semplicità che le parentesi siano esplicitate

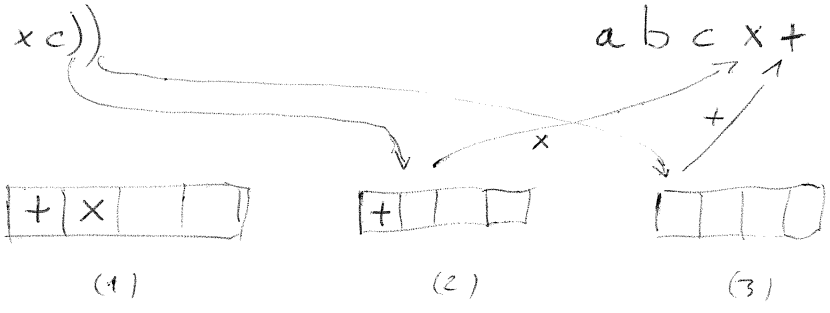
Espressione (minimale): operando

Espressione: coppia di parentesi al cui interno si sono due espressioni separate da un operatore.

La trasformazione avviene leggendo l'espressione infissa da sinistra a destra e applicando le seguenti regole

- parentesi aperta: viene ignorata
- operando: viene accodato direttamente in fondo all'espressione postfissa in costruzione, senza passare per la pila
- operatore: viene eseguita una PUSH di tale operatore
- parentesi chiusa: viene eseguita la POP per riprendere l'operatore, che viene accodato in fondo all'espressione postfissa in costruzione.

Es. $(a + (b \times c))$ $\longrightarrow$ a b c x +



In realtà non è necessario IMPORRE le parentesi nell'espressione infissa prima della conversione.

Supponiamo soltanto che sia SINTATTICAMENTE CORRETTA e terminata dal simbolo speciale $. L'esecuzione delle azioni inizia ponendo una copia di $ nella pila vuota e ha termine quando l'espressione infissa diventa vuota.

Regole di conversione:

- se il simbolo è un operando, viene accodato direttamente in fondo all'espressione postfissa in costruzione
- se il simbolo è un operatore, una parentesi, o il simbolo $, si applica una tra 4 regole, secondo il progetto che segue

CIMA PILA ↓ \ SIMBOLO CORRENTE →	$	+ -	x /	(	)
$	4	1	1	1	
+ -	2	2	1	1	2
x /	2	2	2	1	2
(		1	1	1	3

1. PUSH del simbolo corrente e passaggio al successivo
2. POP. L'operatore ottenuto viene accodato in fondo all'espressione in costruzione
3. Ignorato il simbolo corrente. POP. Ignorato il simbolo restituito da POP. Passaggio al simbolo successivo
4. La conversione ha avuto termine. Il simbolo corrente viene cancellato dall'espressione infissa. POP. Ignorato il simbolo restituito da POP.

Es. $2 \times (6 + 5)$

Supponiamo che la sequenza sia memorizzata in un array: ogni elemento occupa un elemento dell'array.

Array

2	x	(	6	+	5	)	$
0	1	2	3	4	5	6	7

Pila

$					

CONFIGURAZIONE INIZIALE

$\underline{2}\ x\ (\ 6\ +\ 5\ )\ \$$ — 2

$2\ \underline{x}\ (\ 6\ +\ 5\ )\ \$$ — 1. | $ | x | | | |

$2\ x\ \underline{(}\ 6\ +\ 5\ )\ \$$ — 1. | $ | x | (| | |

$2\ x\ (\ \underline{6}\ +\ 5\ )\ \$$ — 2 6

$2\ x\ (\ 6\ \underline{+}\ 5\ )\ \$$ — 1. | $ | x | (| + | |

$2\ x\ (\ 6\ +\ \underline{5}\ )\ \$$ — 2 6 5

$2\ x\ (\ 6\ +\ 5\ \underline{)}\ \$$ — 2. | $ | x | (| | |

2 6 5 +

$2\ x\ (\ 6\ +\ 5\ )\ \underline{\$}$ — 3. | $ | x | | | |

$2\ x\ (\ 6\ +\ 5\ )\ \underline{\$}$ — 2. | $ | | | | |

2 6 5 + x

$2\ x\ (\ 6\ +\ 5\ )\ \underline{\$}$ — 5. | | | | | |

$2\ x\ (\ 6\ +\ 5\ )$

Risultato conversione: 2 6 5 + x

Teorema 2.1 L'algoritmo di conversione da notazione infissa a postfissa è CORRETTO.

Dimostrazione. Consideriamo l'espressione infissa $e = e_1 \, a \, e_2$ dove a è un operatore ed e_1 ed e_2 espressioni infisse.

La proprietà chiave dell'algoritmo è che nel momento in cui a viene messo nella pila, nessun operatore di e_1 è nella pila.

> Nota: nell'esempio considerato $2 * (6 + 4)$ la proprietà è banalmente soddisfatta essendo $e_1 = 2$. Per rendersene conto si consideri un altro esempio, tipo $(2 + 3) * (5 + 1)$.

Guardando infatti la tabella delle operazioni si rende conto che un operatore viene inserito in pila quando sulla cima si trova \$ o (, oppure un operatore di priorità minore (appartenente ad e_2).

Negli altri casi, in cui sulla cima della pila c'è un operatore di e_1 e il simbolo che si sta considerando è l'operatore a, viene eseguita la regola 2. che è una POP. Cioè, l'operatore di e_1 viene tolto dalla pila e accodato all'espressione postfissa in costruzione. Solo quando tutti gli operatori di e_1 sono fuori pila a viene inserito.

Convinti di ciò, procediamo per induzione sul numero $|e| = n$ di operatori dell'espressione e, che, quando tutti gli operatori di e sono stati estratti dalla pila, l'algoritmo ha prodotto in output la versione postfissa $p(e)$ dell'espressione infissa e.

Base dell'induzione. $n = 0$. Un solo operando, appeso senza passare per la pila all'espressione postfissa in costruzione.

Ipotesi induttiva. $n > 0$. Quando si analizza d, e viene messo in pila, tutti gli $|e_1| < n$ operatori di e_1 sono stati estratti dalla pila e $p(e_1)$ è stata data in output.

Passo induttivo. L'operatore d viene messo in pila e inizia la scansione di e_2. Gli operatori di e_2 vengono messi in pila. Quando si raggiunge il simbolo $ o), cioè alla fine di e_2, vengono eseguite una serie di applicazioni della regola 2. (POP e accodamento all'espressione in costruzione)

Poiché $|e_2| < n$, per ipotesi induttiva, $p(e_2)$ viene accodata all'output.

Infine, sulla cima della pila troviamo d, che verrà estratto e accodato alla sequenza postfissa in costruzione producendo

$$p(e_1)\, p(e_2)\, d$$

■