Pile

Le pile: specifiche e realizzazioni attraverso rappresentazioni sequenziali e collegate. Pile e procedure ricorsive

Algoritmi e Strutture Dati + Lab
A.A. 18/19

Informatica
Università degli Studi di Bari “Aldo Moro”

Nicola Di Mauro

Specifica sintattica

- Una pila è una sequenza di elementi di un certo tipo in cui è possibile aggiungere o togliere elementi solo da un estremo della sequenza (la “testa”).
- Può essere vista come un caso speciale di lista in cui l’ultimo elemento inserito è il primo ad essere rimosso (lifo) e non è possibile accedere ad alcun elemento che non sia quello in testa.

Specifica sintattica

- tipi: pila, boolean, tipoelem
- Operatori:
  - creapila: ( ) → pila
  - pilavuota: (pila) → boolean
  - leggipila: (pila) → tipoelem
  - fuoripila: (pila) → pila
  - inpila: (tipoelem, pila) → pila

Specifica semantica

- Tipi:
  - pila=insieme delle sequenze P=<a1, a2, …, an> , n >=0, di elementi di tipo tipoelem gestita con accesso LIFO;
  - boolean=insieme dei valori di verità.
- Operatori:
  - creapila=p
    - post: p=<> (la sequenza vuota)
  - pilavuota(p)=b — empty
    - post: b=vero se p= <>, b=falso altrimenti
  - leggipila(p)=a — top
    - pre: p=<a1, a2, …, an > n>=1
    - post: a=a1
  - fuoripila(p)=p' — pop
    - pre: p=<a1, a2, …, an > n>=1
    - post: p'=<a2, a3, …, an > se n>1 p'= <> se n=1
  - inpila(a, p)= p' — push
    - pre: p=<a1, a2, …, an > n>=0
    - post: p'=<a, a1, a2, …, an >

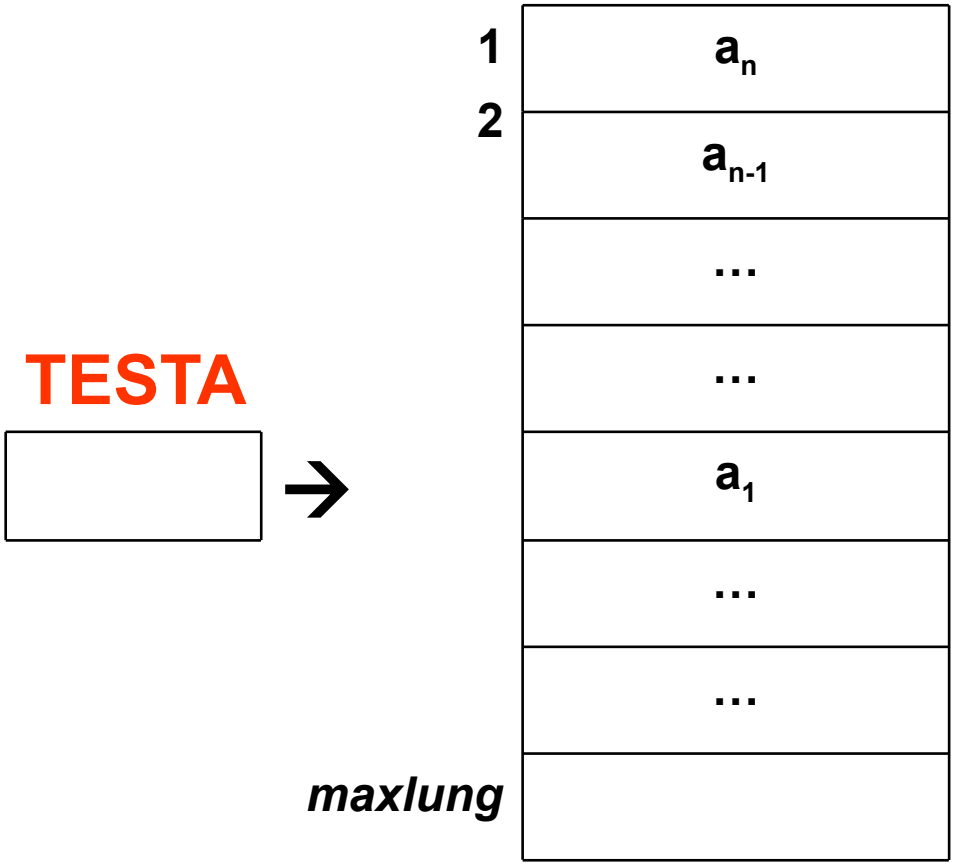
Realizzazioni

- La pila è un caso particolare di lista e ogni realizzazione descritta per la lista funziona anche per la pila. Possiamo definire la corrispondenza tra gli operatori.

- creapila() → crealista()
- pilavuota(p) → listavuota(p)
- leggipila(p) → leggilista(primolista(p),p)
- fuoripila(p) → canclista(primolista(p),p)
- inpila(a,p) → inslista(a,primolista(p),p)

Realizzazione con vettore

- Vanno memorizzati gli n elementi della pila, in ordine inverso, nelle prime n posizioni del vettore, mantenendo un cursore alla testa della pila.

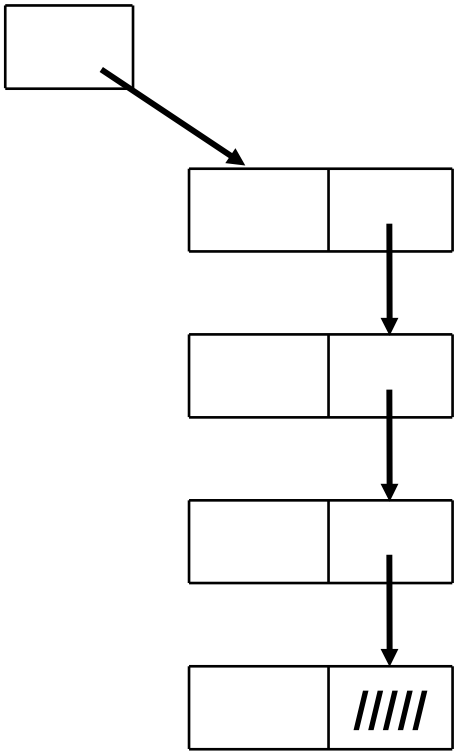


Realizzazione con vettore /2

- Con questa realizzazione, ogni operatore richiede tempo costante per essere eseguito.
- La rappresentazione mediante array presenta due svantaggi:
  - richiede di determinare a priori un limite al numero massimo di elementi della pila;
  - lo spazio di memoria utilizzato è indipendente dal numero effettivo di elementi.
- Per contro:
  - al contrario delle liste, gli inserimenti e le cancellazioni non richiedono spostamenti perché effettuati ad una estremità dell'array

Realizzazione con puntatori

- Ci riferiamo alla pila con un puntatore alla cella che si trova in cima



Pile e procedure ricorsive

- Una delle applicazioni più interessanti delle pile riguarda l'esecuzione di programmi ricorsivi.
- L'esecuzione di una procedura ricorsiva prevede il salvataggio dei dati su cui lavora la procedura al momento della chiamata ricorsiva.
  - Tali dati vengono “ripristinati” quando la computazione “interna” termina (meccanismo lifo).
- Le diverse chiamate attive sono organizzate in una pila.
  - La chiamata più recente è quella che si conclude per prima
- Nella pila vanno salvati i parametri (e le eventuali variabili locali), il punto di ritorno, cioè l’etichetta della istruzione da cui ripartire al termine della computazione “interna”

Pile e procedure ricorsive /2

- Grazie alle pile è sempre possibile, dato un programma ricorsivo, trasformarlo in uno iterativo.
- Come è noto, i programmi ricorsivi corrispondono a metodi solutivi particolarmente adatti a problemi per i quali:
  - la soluzione del problema di rango n è definibile in termini della soluzione del problema di rango inferiore a n;
  - è definibile una soluzione per assioma sul problema di rango minimo (1 o 0).
- Esempio
  - la successione di fibonacci è definita come una successione il cui k-esimo elemento è uguale alla somma dei due che lo precedono:

    fib(1)=1 fib(2)=1 livello assiomatico o di base
    fib(k)=fib(k-1)+fib(k-2) per k>2

Pile e procedure ricorsive /3

- La sequenza di fibonacci si può creare mediante una funzione ricorsiva che calcola il k-esimo numero della successione

```
int fib(int k)
    if (k=1) or (k=2) then f = 1
    else f = fib(k-1)+fib(k-2)
    return f
```

- Gli stati della pila per k=4
  - fib(4)
  - fib(3), fib(4)
  - fib(2), fib(3), fib(4)
  - fib(1), fib(3), fib(4)

Ricorsione e strutture dati

- La ricorsione può essere usata per formalizzare un'ampia classe di strutture di dati che mostrano caratteristiche ricorsive nella struttura.
- In generale, se intendiamo per sequenza di dati un aggregato in cui sia riconoscibile un primo elemento, un secondo… un successivo, possiamo definirla ricorsivamente come:
  - un aggregato di dati eventualmente vuoto; livello assiomatico
  - un aggregato non vuoto, in cui è individuato un primo elemento che insieme ai successivi (aggregato) costituisce ancora una sequenza.
- Se indichiamo la sequenza di n elementi con $s_n=a_1, a_2, \ldots, a_n$ allora s_n sarà:
  - per n=0 $s_0=\{\}$ livello assiomatico
  - per n>0 $S_n = \{s_{n-1}, a_n\}$

La torre di Hanoi

- In un monastero tibetano ci sono n dischi d'oro, forati al centro, tutti di diametro diverso. I dischi sono infilati in un piolo verticale, accatastati per diametro decrescente a partire dal basso. I monaci devono spostare tutti i dischi dal piolo in cui si trovano in un altro piolo, formando una catasta identica a quella di partenza, spostando un disco alla volta senza mai sovrapporre un disco più grande su un disco più piccolo. È possibile usare un terzo piolo di appoggio per effettuare trasferimenti.
- Regole
  - si può spostare un solo disco per volta da un piolo ad un altro;
  - dopo ogni mossa, per ogni piolo, i dischi devono avere diametro decrescente dal basso verso l'alto.

La torre di Hanoi /2

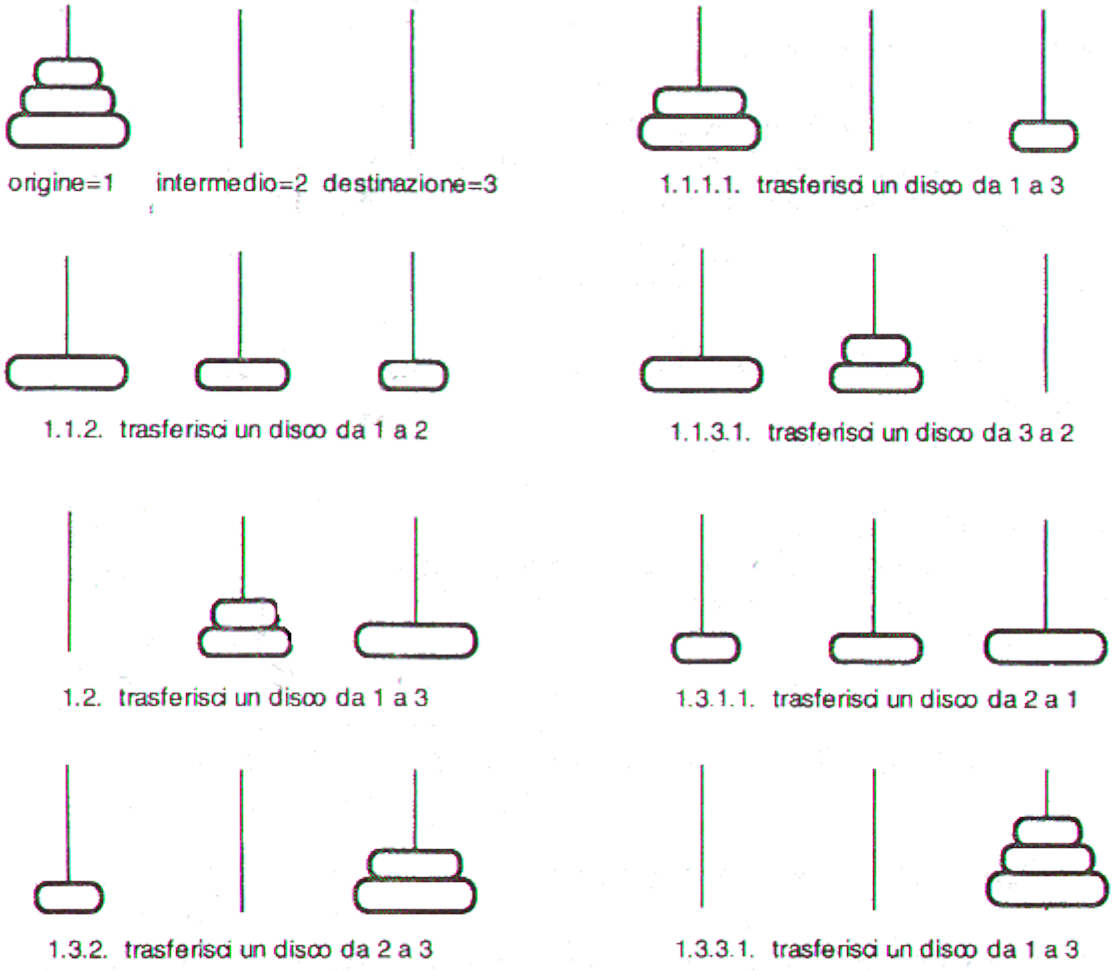


Figura 3.5: Sequenza di trasferimenti di dischi eseguiti dalla procedura ricorsiva TORRIDIHANOI per $n = 3$.

La torre di Hanoi /3

- Per dire spostare n dischi dal piolo sorgente al piolo destinazione usando il piolo ausiliario usiamo la notazione
  - p(ndischi, sorgente, ausiliario, destinazione)
- La formalizzazione della torre di hanoi di ordine n, cioè spostare n dischi dal piolo A al piolo C, usando il piolo B, può essere:
  - p(n; A, B, C)
- È possibile utilizzare la scomposizione ricorsiva: spostare n-1 dischi da A a B usando C come ausiliario e spostare 1 disco da A a C usando B come ausiliario
- La scomposizione di ordine n è riportabile nei termini dello stesso problema di ordine n-1 e lo spostamento di un solo disco costituisce il livello assiomatico.
  - p(n-1; A, C, B)
  - p(1; A, B, C)
  - p(n-1; B, A, C)

La torre di Hanoi /4

- Nel caso di n=3 i sottoproblemi sono
  - P1: p(2, A, C, B) decomponibile in
    - P11: p(1, A, B, C)
    - P12: p(1, A, C, B)
    - P13: p(1, C, A, B)
  - P2: p(1, A, B, C)
  - P3: p(2, B, A, C)
    - P31: p(1, B, C, A)
    - P32: p(1, B, A, C)
    - P33: p(1, A, B, C)

La torre di Hanoi /5

- Strategia
  - a) muovi gli n-1 dischi dal piolo a sinistra al piolo di mezzo usando come piolo ausiliario quello di destra;
  - b) muovi il disco che rimane dal piolo di sinistra a quello di destra;
  - c) muovi gli n-1 dischi dal piolo di centro a quello di destra usando come piolo ausiliario quello di sinistra.

```
move(int n, pole sorgente, pole ausiliario, pole destinazione)
    if n=1 then
        muovi_un_disco_da_sorgente_a_destinazione
    else
        move(n-1, sorgente, destinazione, ausiliario)
        muovi_un_disco_da_sorgente_a_destinazione
        move(n-1, ausiliario, sorgente, destinazione)
```

La torre di Hanoi /6

```
torre()
  definizione di tipi:
   pole: enumerativo con valori {sinistra, centro,destra}
   interopos: intero maggiore di 0


   cin >> numerodischi
   cout << “per ” << numerodischi << “ dischi i movimenti  richiesti sono:”
   move(numerodischi, sinistra, centro, destra)

printpole(pole p)
   case p of
      sinistra: cout << “sinistra”
      centro:   cout << “centro”
      destra:   cout << “destra”

muovisorg_a_destin()
   cout << “muovi un disco da”
   printpole(sorgente)
   cout << “a”
   printpole(destinazione)
   cout << “usando come ausiliario”
   printpole(ausiliario)
```

La torre di Hanoi /7

```
move(interopos n,pole sorgente, pole ausiliario, pole destinazione)
  if n=1 then
      muovisorg_a_destin
   else
      move(n-1, sorgente, destinazione, ausiliario)
      muovisorg_a_destin
      move(n-1, ausiliario, sorgente, destinazione)
```

La torre di Hanoi: versione iterativa

- Per trasformare una procedura ricorsiva in una iterativa bisogna
  - a) creare una pila dopo il begin iniziale;
  - b) sostituire ogni chiamata ricorsiva con una sequenza di istruzioni che:
    - salvano nella pila i valori dei parametri delle variabili locali e l'etichetta della istruzione seguente alla chiamata ricorsiva;
    - assegnano ai parametri gli opportuni valori;
    - effettuano un salto all'istruzione che segue la creazione della pila
  - c) introdurre prima dell'end finale istruzioni che, nel caso la pila non sia vuota, estraggono dalla pila i valori salvati e saltano alla istruzione la cui etichetta è uguale al punto di ritorno.

La torre di Hanoi: versione iterativa /2

- Tali regole sono valide se i parametri si intendono passati per valore.
- Quando parametri e variabili locali sono di tipo diverso nella pila è conveniente memorizzare un record di attivazione che contiene lo stato della computazione sospesa.

La torre di Hanoi: versione iterativa /3

Definizione di tipi:
    tipoelem: elemento strutturato con componenti
    numerodischi: interopos
    piolorig, piolodest, pioloaus: pole
    ritorno: intero

```
move(int n; pole sorgente, pole ausiliario, pole destinazione)
    creapila(s)
1:  if n=1 then
        muovisorg_a_destin
        goto 3
    stato.numerodischi = n
    stato.piolorig  = sorgente
    stato.pioloaus = ausiliario
    stato.piolodest = destinazione
    ritorno = 2
    inpila(stato,s)
    n = n-1
    temp = destinazione
    destinazione = ausiliario
    ausiliario = temp
    goto 1
```

La torre di Hanoi: versione iterativa /4

```
2:  muovisorg_a_destin
    stato.numerodischi = n
    stato.piolorig = sorgente
    stato.pioloaus = ausiliario
    stato.piolodest = destinazione
    stato.ritorno = 3
    inpila(stato,s)
    n = n-1
    temp = sorgente
    sorgente = ausiliario
    ausiliario = temp
    goto 1
3:  if not pilavuota(s) then
        stato = leggipila(s)
        fuoripila(s)
        n = stato.numerodischi
        sorgente = stato.piolorig
        destinazione = stato.piolodest
        ausiliario = stato.pioloaus
        case stato.ritorno of
            2: goto 2
            3: goto 3
```