Corso di Informatica

Modulo 4

4 - Strutture astratte lineari

Prerequisiti

- Tecnica della programmazione
- Strutture dati concrete
- Strutture dati astratte (sequenza, pila, coda)

Introduzione

In questa Unità riprendiamo il concetto di **ADT** e vediamo alcune implementazioni delle strutture astratte notevoli.

Argomenti rev

- ADT
- Definire una ADT
- Implementare una ADT
- Un esempio di ADT
- Caratteristiche di una ADT
- ADT lineari
- Definizione di Sequenza
- ADT Sequenza
- Operazioni sulla Sequenza
- Definizione di Pila
- ADT Pila

Definire una ADT

Per risolvere un dato problema può essere necessario usare e comporre i dati concreti per formare una **struttura astratta di dati** o **ADT**

In generale **definire una ADT** (*Abstract Data Type*) significa:

- definire tipo dei dati da gestire (**dominio**)
- definire le **operazioni** consentite sui dati della ADT (ossia estendere le potenzialità del linguaggio di programmazione a disposizione, creando per esso nuovi tipi e/o nuove operazioni su essi)

Implementare una ADT

Implementare una ADT invece significa

- Dichiarare una struttura opportuna per i dati
- Implementare le operazioni per elaborare i dati della ADT
- Produrre una documentazione contenente
  - il nome del nuovo tipo di dato creato
  - i nomi delle operazioni (funzioni) realizzate, con le loro interfacce di chiamata (ordine, numero e tipo dei parametri)

Nella programmazione ad oggetti, il concetto di classe è un esempio evidente di ADT.

ADT lineari

In questa Unità, vogliamo studiare alcuni modelli di ADT, dette **strutture astratte lineari**, che possono essere adattati a molte situazioni reali.

- **ADT Sequenza**
- **ADT Pila**
- **ADT Coda**

Sono dette **lineari** poiché ogni dato presenti in esse ha un unico precedente ed un unico successivo (eccettuati il primo e l'ultimo).

Definizione di Sequenza

La sequenza, può essere definita iterativamente o ricorsivamente:

DEFINIZIONE ITERATIVA:

Una **sequenza s(T)** è un insieme di informazioni, di un dato tipo T, disposte **consecutivamente** che hanno, ad eccezione della prima e dell'ultima, un predecessore ed un successore

DEFINIZIONE RICORSIVA:

Una **sequenza s(T)** è un insieme vuoto (Ø) di informazioni, di un dato tipo T, oppure una informazione di tipo T avente come successore una **sequenza s(T)**.

ADT Sequenza

L'**ADT Sequenza** rappresenta una struttura astratta formata da una successione teoricamente illimitata di nodi o celle consecutivi contenenti dati di un tipo T, che indicheremo con:

$$s(T)=[a_1, a_2,\ldots a_n] \qquad a_i \in T,\ i=1,2,\ldots n$$

Indichiamo con

- n la lunghezza della sequenza s(T)
- i singoli a_i gli elementi che formano la sequenza
- a_1 il valore di **testa** di s(T)
- a_n il valore di **coda** di s(T)
- s=Ø la sequenza vuota

Operazioni sulla Sequenza

Le operazioni su s possono essere definite ponendo:

S = {insieme di tutte le possibili sequenze s}

N = {insieme dei naturali n}

T = {tipo dei dati a_i}

- **Creazione**: inizializzazione sequenza
- **Inserimento**: inserimento elemento di tipo T in s
- **Eliminazione**: elimina un elemento da s
- **Ricerca**: ricerca di un elemento nella sequenza
- **Test sequenza vuota**: vero/falso se s è vuota/non vuota
- **Lunghezza**: numero di elementi presenti in s

Implementazione Sequenza

Possiamo considerare:

- **Implementazione statica**
- **Implementazione dinamica**.

La prima usa un array, la seconda, invece, usa liste e puntatori.

Ciascuna delle due implementazioni, quindi, presenta i vantaggi e gli svantaggi relativi al tipo di struttura dati utilizzata.

Implementazione statica

Possiamo considerare la seguente struttura dati:

```
const int MAX = 10;
typedef struct
{   int conta;
    int dati[MAX];
} sequenza;

sequenza s;
```

Implementazione statica

Prototipi:

```
void svuota (sequenza &s);                    // svuota la sequenza;
int inserisci (int n, int pos, sequenza &s); // 0=successo,
                                              //-1=insuccesso (piena o op. illegale)
void stampa (sequenza s);
int elimina (int pos, sequenza &s);           // 0=successo, -1=vuota
bool vuota (sequenza s);                      // true=vuota, false=non vuota
int esiste (int n, sequenza s);               // ricerca sequenziale -1=insuccesso
int ricerca (sequenza s, int valore)
```


Implementazione statica

```
void svuota (sequenza &s);
{
  int i;
  s.conta=0;
  for (i=0; i<MAX; i++)
    s.dati[i]=0;
}
```

Si osservi il parametro *s* passato per indirizzo, che la funzione può così modificare.

Implementazione statica

```
int inserisci (int n, int pos, sequenza &s)
{
  if ((pos>=1) && (pos<=MAX))
  {
    s.dati[pos-1]=n;
    s.conta++;
    return 0;
  }
  else
    return -1;
}
```

La funzione riceve il parametro *s*, passato per indirizzo, la posizione *pos* in cui inserire e il valore *n* da inserire.

La funzione restituisce -1 in caso di insuccesso (op. illegale), 0 in caso di successo.

Implementazione statica

```
void stampa (sequenza s)
{
  int i;
  cout<<"[";
  for (i=0; i<MAX-1; i++)
    cout<<s.dati[i]<<"-";
  cout<<s.dati[MAX-1]<<"]"<<endl;
}
```

La funzione riceve il parametro *s*, passato per valore, e stampa i valori presenti nella sequenza con il formato seguente:

[a1-a2-a3….-an]

Implementazione statica

```
int elimina (int pos, sequenza &s)
{
  if ((pos<0) || (pos>MAX)) return -1;
  if (vuota(s)) return -1;
  s.dati[pos-1]=0;
  s.conta--;
  return 0;
}
```

La funzione riceve il parametro *s*, passato per idirizzo, e la posizione *pos* da eliminare.

L'elemento viene considerato eliminato se posto a valore 0.

Implementazione statica

```
bool vuota (sequenza s)
{
  if (s.conta==0)
    return true;
  else
    return false;
}
```

La funzione riceve il parametro *s*, passato per valore, e svuota la sequenza semplicemente ponendo a 0 il campo *conta*.

Si tratta di un esempio di **cancellazione logica**.

Implementazione statica

```
int ricerca (sequenza s, int valore)
{
   int i=0;
   while ((i<MAX) && (s.dati[i]!=valore))
     i++;
   if (i==MAX)
     return -1;
   else
     return i+1;
}
```

La funzione riceve il parametro *s*, passato per valore, e l'elemento da cercare. L'esito della funzione viene valutato come -1 (insuccesso) oppure come posizione dell'elemento eventualmente trovato.

Implementazione dinamica

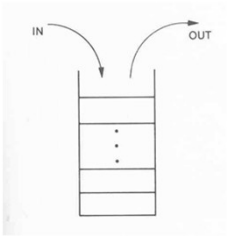
IMPLEMENTAZIONE DINAMICA DELLA SEQUENZA

Definizione di Pila

La **pila** può essere immaginata come una sequenza in cui supponiamo però che **gli inserimenti e le eliminazioni possano essere fatti esclusivamente ad una sola delle due estremità**.

Per quanto detto, la pila (o *stack*) con i vincoli imposti alle operazioni, consente l'eliminazione degli elementi in ordine inverso a quello di immissione.

Per questo motivo si dice che la pila è una struttura con disciplina *Last In First Out o* **LIFO;** l'ultimo oggetto immesso, sarà il primo ad essere estratto.



Definizione di Pila

RIVEDERE

Un esempio di gestione secondo la tecnica LIFO è un gruppo di oggetti incolonnati su un piano (da cui il nome **pila**), per i quali il prelievo o l'inserimento vengono fatti dalla cima.

Un'applicazione informatica del concetto di pila è stata vista trattando i sottprogrammi; quando in un programma P si presenta l'istanza di un sottoprogramma SP, l'elaborazione di P viene temporaneamente sospesa in favore di SP. La **pila di attivazione** mantiene l'indirizzo (nello **stack pointer**) della posizione da cui dovrà riprendere l'esecuzione di P.

ADT Pila

L'**ADT Pila** rappresenta una struttura astratta formata da una successione teoricamente illimitata di nodi o celle consecutivi contenenti dati di un tipo T, che indicheremo con:

$$p(T)=[a_1, a_2,\ldots a_n] \qquad a_i \in T,\ i=1,2,\ldots n$$

Indichiamo con

- n la lunghezza della pila p(T)
- i singoli a_i gli elementi che formano la pila
- a_1 il valore di **cima** di p(T)
- s=Ø la pila vuota

Operazioni sulla Pila

Le operazioni su s possono essere definite ponendo:

P = {insieme di tutte le possibili pile p}

T = {tipo dei dati a_i}

- **Creazione**: inizializzazione pila
- **Inserimento**: inserimento elemento di tipo T in p
- **Eliminazione**: elimina un elemento da p
- **Test pila vuota**: vero/falso se p è vuota/non vuota
- **Interrogazione**: valore dell'elemento in cima a p

Implementazione Pila

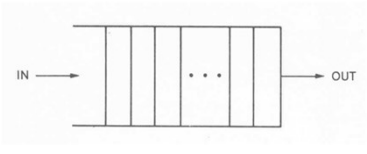
La **creazione**

Definizione di Coda

La **coda** è immaginabile come una sequenza caratterizzata però dal fatto che **gli inserimenti avvengono esclusivamente ad una estremità e le eliminazioni all'altra.**

Per quanto detto, la coda (o *queue*) con i vincoli imposti alle operazioni, consente l'eliminazione degli elementi in base all'ordine di immissione.

Per questo motivo si dice che la coda è una struttura con disciplina *First In First Out o* **FIFO;** il primo oggetto immesso, sarà il primo ad essere estratto.



ADT Coda

RIVEDERE

Esempi di informazioni gestite secondo la tecnica FIFO sono un gruppo di veicoli presenti alla porta di un casello autostradale oppure una fila di persone in attesa di essere chiamate, per cui il prelievo e l'inserimento avvengono alle estremità opposte.

Un'applicazione informatica del concetto di coda può essere il buffer di una stampante, i processi in attesa della CPU o o una serie di dati da elaborare in ordine cronologico.

ADT Coda

L'**ADT Coda** rappresenta una struttura astratta formata da una successione di nodi o celle consecutivi contenenti dati di un tipo T, che indicheremo con:

$$c(T)=[a_1, a_2,\ldots a_n] \qquad a_i \in T,\ i=1,2,\ldots n$$

Indichiamo con

- n la lunghezza della coda c(T)
- i singoli a_i gli elementi che formano la coda
- a_1 il valore di **testa** di c(T)
- a_n il valore di **coda** di c(T)
- c=Ø la coda vuota

Operazioni sulla Coda

Le operazioni su s possono essere definite ponendo:

C = {insieme di tutte le possibili code c}

T = {tipo dei dati t}

- **Creazione**: inizializzazione coda
- **Inserimento**: inserimento elemento di tipo T in c
- **Eliminazione**: elimina un elemento da c
- **Test pila vuota**: vero/falso se c è vuota/non vuota
- **Interrogazione**: valore dell'elemento di coda di c

Implementazione Coda

La **creazione**

Cosa ho imparato

- Tecniche per implementare strutture astratte
- Strutture dati che implementano strutture astratte

Cosa ho imparato a fare

- Creare modelli che rappresentino strutture astratte
- Progettare l'implementazione di strutture astratte
- Scrivere codice per implementare strutture astratte

Terminologia

- Implementare una ADT

Riepilogo

In questa Unità abbiamo visto cosa si intende per **ADT**: una **struttura astratta di dati** che gode delle proprietà tipiche delle classi in OOP. I **dati** in essa presenti sono incapsulati e possono essere manipolati soltanto mediante le **operazioni** tipiche della ADT fornite all'utente.

Abbiamo visto una prima serie di strutture dati astratte, dette **strutture astratte lineari**: in queste strutture, ogni elemento, ad eccezione del primo e dell'ultimo, possiede un unico precedente ed un unico successivo.

Le ADT lineari viste sono: **Sequenza**, **Pila** e **Coda**.

Riepilogo

Per ciascuna ADT abbiamo dato un set di **operazioni base** che consentono di gestire i dati presenti nella struttura.

La **Sequenza** è l'ADT che consente operazioni di **inserimento** (in testa, in coda e intermedio) di **eliminazione** (in testa, in coda e intermedio), di **ricerca**, di **test sequenza vuota** e di **creazione**.

La ADT **Pila**, o *stack*, prevede le operazioni di **creazione**, di **interrogazione**, di **test pila vuota**, di **creazione**; le operazioni di **immissione** e di **estrazione** avvengono nella stessa estremità e sono dette, rispettivamente, Push() e Pop(). La pila segue una disciplina detta **LIFO**, per cui l'ordine di estrazione segue l'ordine inverso rispetto a quella di immissione.

Riepilogo

La ADT **Coda**, o *queue*, prevede le operazioni di **creazione**, di **interrogazione**, di **test coda vuota**, di **creazione**; le operazioni di **immissione** e di **estrazione** avvengono alle estremità opposte; la coda segue una disciplina detta **FIFO**, per cui l'ordine di estrazione segue lo stesso ordine di immissione.

Altre fonti di informazione

- P.Gallo, F.Salerno – Informatica Generale vol. 2 – Minerva Italica
- G. Callegarin – Corso di Informatica – vol. 2 – CEDAM
- F. Luccio – Strutture, linguaggi, sintassi – Boringhieri
- A.Garavaglia, F.Petracchi, F.Forte-Informatica . Vol. 2 - Masson