Fondamenti di Informatica e Laboratorio T-AB
T-15 – Strutture dati

Paolo Torroni

Dipartimento di Elettronica, Informatica e Sistemistica
Università degli Studi di Bologna

Anno Accademico 2008/2009

Sommario

Strutture astratte

- Il progetto delle strutture dati è una parte della soluzione del problema che si vuole risolvere.
- Una **struttura dati astratta** definisce l'organizzazione delle informazioni indipendentemente dalla traduzione in uno specifico linguaggio di programmazione (*implementazione*).
- Alcune strutture sono utilizzate comunemente per molti problemi:
  - liste;
  - pile (stack);
  - code (queue);
  - contenitori (container);
  - code con priorità (priority queue).
  - insiemi (set) e multiset;
- Importante conoscere il funzionamento di queste strutture, e saperle implementare.

Problemi dei tipi di dato visti finora

- Alcuni tipi di dato messi a disposizione dal C:
  - array;
  - `struct`;
  - puntatori;
- Quali possono essere i limiti di questi tipi in determinati contesti?
  - occupazione di memoria;
  - velocità di esecuzione;
  - chiarezza della soluzione.

Problemi di occupazione di memoria

- Problema della definizione di array come variabile automatica:
  - Dimensione fisica (F) diversa dalla dimensione logica (L).
  - Se $F < L$, struttura dati sottodimensionata: la soluzione proposta non funziona.
  - Se $F >> L$, eccessivo sovradimensionamento: cattivo utilizzo delle risorse del computer (memoria).
- Esempio: prenotazioni di 700 studenti delle scuole medie relativamente a una delle tre gite scolastiche: Vienna, Parigi, Roma.
  - Posso definire 3 array di stringhe: `Vienna`, `Parigi`, `Roma`;
  - Quanti elementi per ciascun array?
  - Devo sempre far riferimento al caso peggiore.
- Posso usare una variabile dinamica, ma in ogni caso a un certo punto devo fissare il valore di F.

Problemi di velocità di esecuzione

- Ovvero: *performance*, *complessità*, *scalabilità*.
- Esempio: devo gestire un insieme ordinato di $n \leq k$ valori.
  - Uso un array di lunghezza k.
  - Cosa succede quando devo cancellare un elemento?
  - Cosa succede quando devo inserire un elemento?
- In media: devo riscrivere $n/2$ elementi.
- Soluzione *costosa* (in termini di tempo) soprattutto per n grande.

Problemi di chiarezza della soluzione

- Soluzione chiara se l'organizzazione della struttura corrisponde alla logica di utilizzazione delle informazioni in essa contenuta.
- Esempio: voglio mantenere un albero genealogico.
  - Come faccio a mantenere le informazioni su 4 generazioni di una famiglia, usando un array?
    - dimensionamento
    - organizzazione dei dati
    - procedure per l'accesso ai dati (es: individuazione di nuclei familiari/figli/fratelli/etc)
    - inserimento di nuovi dati
    - etc
- Esempio: rappresentazione di una frase in un linguaggio.
  - Come faccio a individuarne i vari elementi? (*parsing*)
  - Come faccio a vedere se appartiene a una certa grammatica?
- Non sempre l'array o la stringa è la soluzione migliore.
- Sarebbe più semplice usare una struttura dati che rappresenta direttamente un albero.

Tipi di dato astratto

- Un **tipo di dato astratto** (**ADT**) è la specifica di un insieme di dati, delle loro proprietà, e delle operazioni che si possono eseguire sui dati.
- Esempio di ADT: `Container`.
  - contiene collezioni di elementi,
  - definisce un **costruttore**, cioè un'operazione per creare un container vuoto;
  - le operazioni che si possono eseguire sui dati sono:
    - `size`: restituisce il numero di elementi contenuti;
    - `clear`: elimina tutti gli elementi;
    - `insert`: inserisce nuovi elementi nel container;
    - `remove`: elimina determinati elementi dal container;
    - `find`: fornisce accesso a un elemento del container.
  - Stack, code, alberti, insiemi etc. sono esempi di `Container`.

Tipi di dato astratto

- Un altro esempio di ADT: `Rational`.
- Con la rappresentazione in virgola mobile non è possibile rappresentare fedelmente tutti i razionali.
- Inoltre i tipi `float` e `double` non sono chiusi rispetto alle operazioni dell'aritmetica classica.
- Si potrebbe definire un ADT `Rational` in cui un numero razionale è rappresentato come il quoziente di due interi a/b.
- L'ADT definisce una interfaccia in cui sono presenti:
  - un costruttore, che crea un'istanza di `Rational` a partire da due interi a e $b \neq 0$
  - le operazioni di somma, sottrazione, prodotto, divisione, elevamento a potenza, confronto, semplificazione, conversione a reali in virgola mobile.
- L'ADT dovrebbe specificare anche proprietà dei dati e precondizioni/postcondizioni delle operazioni (es. $b \neq 0$).

Tipi di dato astratto

- Un altro esempio di ADT: `Stack`
- ha un costruttore che crea un'istanza vuota di `Stack`
  - mette a disposizione solo due operazioni per accedere ai dati:
    - `push`: inserisce un elemento;
    - `pop`: elimina un elemento.
  - La politica di accesso ai dati è LIFO (Last In First Out).
  - Altre possibili operazioni (derivate da `Container`):
    - `isEmpty`: per verificare se lo stack è vuoto;
    - `clear`: per eliminare tutti gli elementi dallo stack;
    - `size`: restituisce il numero di elementi nello stack;

Tipi di dato astratto

- Per definire un tipo di dato astratto occorre definirne le proprietà e le operazioni.
- Lo **scopo** è quello di astrarre da una specifica implementazione, e rendere il dato accessibile solo attraverso una **interfaccia**.
- I punti chiave da tenere a mente sono:
  - L'oggetto dovrebbe essere manipolabile solo tramite le operazioni previste per esso;
  - Non deve essere necessario accedere alla struttura interna dell'oggetto in modo diretto.

Lista

- L'ADT `List` è un tipo di **container**.
- Una possibile definizione:
  - contiene collezioni di elementi omogenei;
  - il costruttore genera una lista vuota;
  - l'interfaccia è data dalle seguenti operazioni:
    - `isEmpty`: verifica se la lista è vuota;
    - `insert`: inserisce un elemento nella lista;
    - `head`: restituisce il primo elemento nella lista;
    - `tail`: restituisce una lista contenente tutti gli elementi eccetto il primo;
    - `clear`: elimina tutti gli elementi dalla lista;

Lista

- Potevamo definire l'interfaccia in altri modi:
  - Con proprietà aggiuntive:
    - Es, la lista è ordinata
  - Con operazioni aggiuntive o alternative:
    - `size`: restituisce il numero di elementi nella lista;
    - `max_size`: restituisce il numero massimo di elementi nella lista;
    - `append`: concatena due liste;
    - `insertAt`: inserisce un elemento in una determinata posizione;
    - `delete`: rimuove un elemento dalla lista;
    - `find`: fornisce un riferimento a un elemento della lista.
    - `view`: mostra tutti gli elementi della lista.

Lista

- Vediamo due possibili implementazioni:
  - mediante **array**;
  - mediante **lista collegata**.

Implementazione della lista mediante array

- Occorre assumere una **dimensione fisica** (massima) N;
- Il **costruttore** definisce un array di N elementi, in modo statico o dinamico, e restituisce un riferimento al primo elemento;
- Il riferimento a un elemento specifico della lista viene effettuato tramite un **indice** intero $i \in [0..N]$.
- Bisogna tenere traccia della **dimensione logica** dell'array, tramite un altro intero.

```
typedef int elem; // esempio con lista di interi

typedef struct {
    int N,M; // dimensione fisica (N) e logica (M)
    elem *V; // vettore di elementi
} List;
```

Implementazione della lista mediante array

- Il **costruttore** definisce un array di N elementi, in modo statico o dinamico, e restituisce un riferimento al primo elemento;

```
// crea una lista nuova di dimensione fisica dim
List *newList( int dim ) {
    elem *V; List *L;
    V=( elem* )malloc( dim*sizeof( elem ) );
    L=( List* )malloc( sizeof( List ) );
    L->V=V;
    L->N=dim;
    L->M=0;
    return L;
}
```

Implementazione della lista mediante array

- Interfaccia completa;

```
List *newList( int );
int isEmpty( List* );
int insert( List*, elem );
int insertAt( List*, elem, int );
elem head( List* );
List *tail( List* );
void clear( List* );

int size( List* );
int max_size( List* );
List *append( List*, List* );
int delete( List*, elem );
int find( List*, elem );
void view( List*, char* );
int next( List*, int );
```

Implementazione della lista mediante lista collegata

- Definizione di un elemento;

```
typedef int elem; // esempio con lista di interi

typedef struct LinkedList {
    elem value; // elemento
    struct LinkedList *next; // puntatore al prossimo
} LinkedList;
```

Confronto tra i due tipi di implementazione

- Flessibilità e occupazione di memoria:
  - Occupazione statica/dinamica
  - Determinazione della lunghezza massima
- Performance: (tempo costante: $\mathcal{O}(1)$, tempo linerare$\mathcal{O}(n)$)
  - Accesso: random ($\mathcal{O}(1)$)/sequenziale ($\mathcal{O}(n)$) ?
  - Inserimento: $\mathcal{O}(1)/\mathcal{O}(n)$?
  - Inserimento con iteratore: $\mathcal{O}(1)/\mathcal{O}(n)$?
  - Ordinamento/mantenimento della lista ordinata
  - Località

Altri ADT implementati con linked list

- Sono spesso implementati mediante liste collegate anche altri ADT:
  - stack
  - code
  - alberi binari