Corso di Programmazione I

Strutture dati astratte

Argomenti

- Strutture dati astratte:
  - Lista
    - Rappresentazione statica con array
    - Rappresentazione con lista dinamica a puntatori
  - Pila
    - Rappresentazione statica con array
    - Rappresentazione con lista dinamica a puntatori
  - Coda
    - Rappresentazione statica con array
    - Rappresentazione con lista dinamica a puntatori
  - Albero
  - Tabella
    - Rappresentazioni statiche con array di record, con matrice, con due array
    - Rappresentazione con lista dinamica a puntatori
- Analisi complessità su sequenze lineari

Riferimenti

- Fondamenti di Informatica vol. II
  - Parte prima, capitolo 2: tutto

ADT lista

- Classificazione
  - Ordinamento
    - Ordinate secondo l'ordine di immissione (in pratica non ordinate)
    - Ordinate secondo una relazione di ordinamento degli elementi contenuti
  - Struttura
    - array
    - liste a puntatori
  - Disposizione elementi
    - lineare
    - gerarchica (alberi)
  - Strategia di accesso
    - non ordinata
    - ordinata
      - FIFO (coda)
      - LIFO (pila)
      - altro criterio
  - Allocazione di memoria
    - statica
    - dinamica

Esempio

LISTA ORDINATA			
nome	**domini**	**semantica**	**notazione**
operazioni			
Start	$\phi \rightarrow L$	Inizializza la lista: produce lista vuota	Start(l)
Inserisci	$L \times E \rightarrow L$	Inserisce un elemento secondo l'ordine definito	Inserisci(l,e)
Elimina	$L \times E \rightarrow L$	Estrae un elemento	Elimina(l,e)
Stampa	$L \rightarrow \phi$	Stampa la lista completa	Stampa(l)
predicati			
Empty	L→booleano	La lista è vuota	Empty(l)
Full	L→booleano	La lista è piena	Full(l)
InList	LxE→booleano	L'elemento è presente nella lista	Inlist(l,e)

Un specifica del tipo astratto in C++

```
typedef int E;
struct Record;
typedef Record* Lista;
//Funzioni
void start(Lista& l);
void inserisci(Lista& l, const E e);
void elimina(Lista& l, const E e);
void stampa(const Lista& l);
void distruggi (Lista&);
//Predicati
bool empty(const Lista& l);
bool full(const Lista& l);
bool inlist(const Lista& l, const E e);
```

La dichiarazione del tipo `Record` non è a rigore utile al cliente della libreria ed, anzi, compromette in parte l'information hiding. In realtà, comunque, questo tipo è completamente inutilizzabile all'esterno della libreria poiché non è definito nel `.h` (ma solo dichiarato). Sarà il `.cpp` a definirlo.

Rappresentazione con array 1/4

- Struttura dati:
  - Array statico con una variabile di tipo intero che mantiene il numero corrente degli elementi nella lista
  - La sequenza degli elementi è data dalla posizione stessa degli elementi dell'array: l'ordinamento logico corrisponde con l'ordinamento fisico

Rappresentazione con array 2/4

- In questo caso la definizione del tipo `Record` all'interno del .cpp potrebbe essere del tipo:

```
struct Record {
  E v[N];
  int nelem;
}
```

Rappresentazione con array 3/4

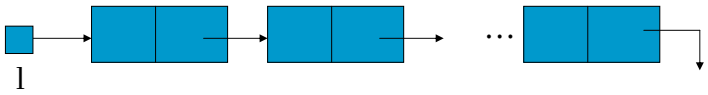
- Lista non ordinata:
  - Inizializzazione (nelem=0)
  - Inserimento in testa (push)
  - Inserimento in coda
  - Ricerca sequenziale
  - Eliminazione in testa (pop)
  - Eliminazione di un elemento dato
  - Analisi dell'elemento di testa (top)
  - Calcolo dei predicati empty e full (nelem==0/nelem==N)

Rappresentazione con array 4/4

- Lista ordinata
  - Le operazioni devono garantire l'ordinamento
  - La gestione è onerosa perché richiede lo spostamento fisico degli elementi in caso di inserimento e eliminazione

Rappresentazione con lista dinamica a puntatori 1/4

- Struttura dati
  - Richiede la dichiarazione di un tipo Record
  - È costituita da un puntatore al tipo Record che rappresenta la testa della lista
- Gli elementi della lista vengono allocati dinamicamente



Rappresentazione con lista dinamica a puntatori 2/4

- In questo caso la definizione del tipo `Record` all'interno del .cpp potrebbe essere del tipo:

```
typedef Record* PRec;
struct Record {
  E elem;
  PRec succ;
};
```

Rappresentazione con lista dinamica a puntatori 3/4

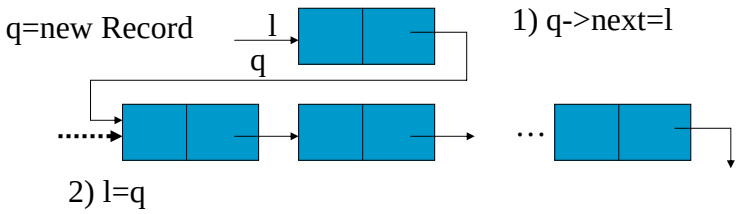
- Lista non ordinata:
  - Inizializzazione (l=0)
  - Inserimento in testa (push)
  - Inserimento in coda
  - Ricerca sequenziale
  - Eliminazione in testa (pop)
  - Eliminazione di un elemento dato
  - Analisi dell'elemento di testa (top)
  - Calcolo dei predicati empty e full (l==0/FALSE)

Rappresentazione con lista dinamica a puntatori 4/4

- Lista ordinata
  - Le operazioni devono garantire l'ordinamento
  - La gestione è molto vantaggiosa rispetto alla soluzione precedente perché non richiede lo spostamento fisico degli elementi in caso di inserimento e eliminazione
- In generale un ulteriore vantaggio è dato dalla occupazione di memoria commisurata all'effettiva necessità e alla possibilità di rilasciare le aree non più utilizzate in seguito alla cancellazione di un elemento

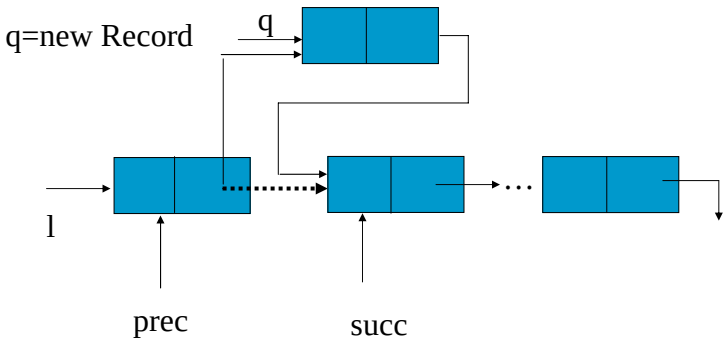
Lista dinamica a puntatori: inserimento (1/2)

Inserimento in testa



Lista dinamica a puntatori: inserimento (2/2)

Inserimento in ordine

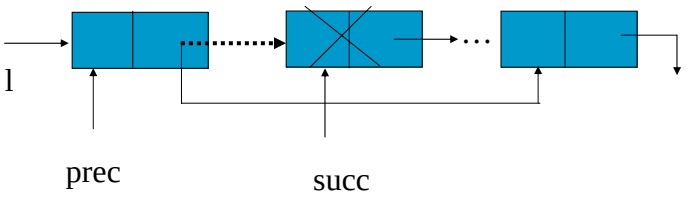


Lista dinamica a puntatori: eliminazione (1/2)

Eliminazione in testa

Lista dinamica a puntatori: eliminazione (2/2)

Eliminazione di un elemento interno alla lista

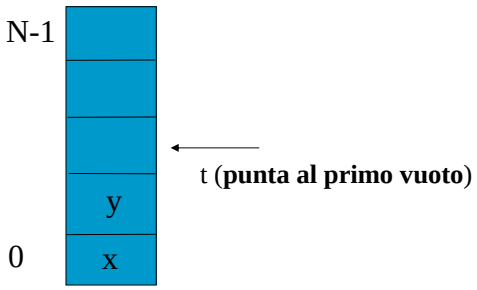


ADT Pila

- PILA: Lista con politica di gestione LIFO: Last In First Out
  - Start
  - Push
  - Pop
  - Top
  - Empty
  - Full
- La struttura dati prevede un indice-puntatore alla testa della pila

ADT Pila: rappresentazione con array

- Start
  - `t=0;`
- Push
  - `p[t]=e;`
  - `t++;`
- Pop
  - `t--;`
  - `e=p[t];`
- Top
  - `e=p[t-1];`
- Empty
  - `t==0;`
- Full
  - `t==N;`

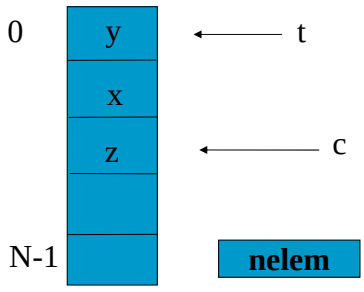


ADT Coda

- CODA: lista con politica di gestione FIFO (First In First Out):
  - Start
  - Append
  - Pop
  - Empty
  - Full
- La struttura dati prevede una variabile contenente il numero degli elementi in coda e sia un indice-puntatore alla testa che un indice-puntatore alla coda
- Una variante molto utilizzata è la coda circolare

ADT Coda: rappresentazione con array 1/2

- L'inserimento avviene in coda
- Il prelievo avviene in testa



ADT Coda: rappresentazione con array 2/2

- Start
  - `nelem=0;`
  - `t=c=0;`
- Append
  - `q[c]=e;`
  - `c=(c+1)%N;`
  - `nelem++;`
- Pop
  - `e=q[t];`
  - `t=(t+1)%N;`
  - `nelem--;`
- Empty
  - `nelem==0;`
- Full
  - `nelem==N;`

L'incremento degli indici-puntatori è effettuato modulo N

Rappresentazione degli ADT Pila e Coda con lista dinamica a puntatori

- È del tutto simile a quanto già visto
- La struttura dati della Pila prevede un puntatore alla testa
- La struttura dati della Coda prevede un puntatore alla testa ed uno alla coda
  - avere a disposizione entrambi i puntatori, sebbene non indispensabile, migliora l'efficienza di alcune operazioni ed è praticamente una scelta sempre adottata
  - ciò migliora, per esempio, l'inserimento in coda poiché non è necessario scorrere tutta la struttura

Alberi

- Definizione (ricorsiva): una struttura ad albero con tipo base T può essere:
  - La struttura vuota
  - Un nodo di tipo T associato ad un numero finito di strutture ad albero disgiunte aventi tipo base T e denominate sottoalberi
- Il nodo da cui si sviluppa l'albero è detto nodo radice
- Un nodo y direttamente collegato sotto il nodo x si dice discendente di x
- Un nodo che non ha discendenti viene chiamato *nodo terminale* o *nodo foglia*
- Il massimo numero di livelli in cui si sviluppa l'albero si dice profondità dell'albero
- Per definizione la radice dell'albero si trova al livello 1

Alberi binari

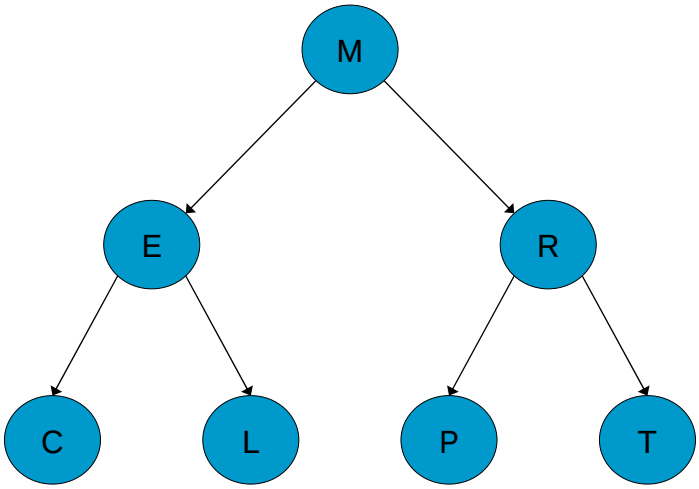
- Il numero di discendenti diretti di un nodo interno è detto grado del nodo
- Il grado di un albero è il massimo grado tra tutti i suoi nodi interni
- Si dice albero binario un albero di grado 2
- Pertanto:
  - Definizione: un albero binario è un insieme finito di nodi che può essere:
    - l'insieme vuoto
    - oppure un nodo radice collegato a due alberi binari disgiunti, detti sottoalbero sinistro e sottoalbero destro
- N.B.:
  - una lista è un albero degenere
  - pertanto la lista è anch'essa una struttura intrinsecamente ricorsiva

Definizioni…

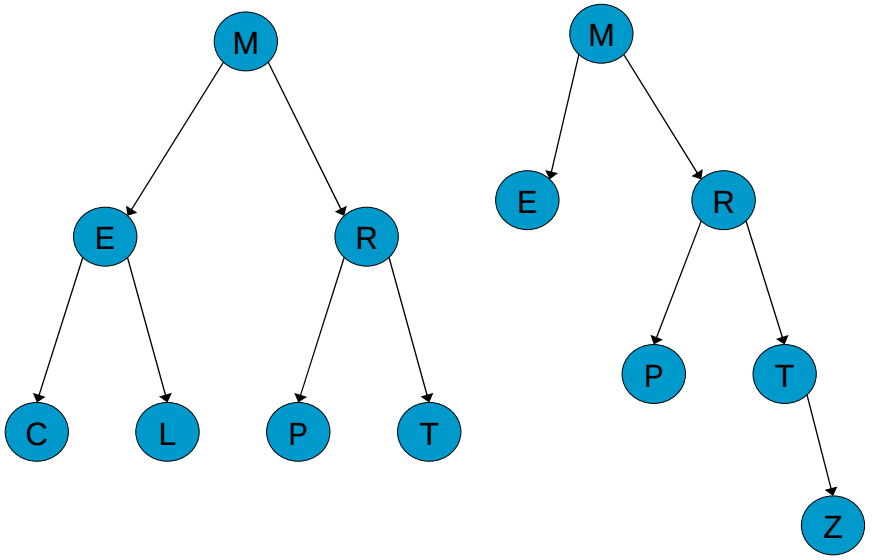
- Un albero è ordinato se e solo se i discendenti di ogni nodo sono ordinati (rispetto ad una chiave)

- i due alberi in figura sono due alberi (diversamente) ordinati.
- In particolare si può definire il seguente ordinamento:
  - Tutti i nodi del sottoalbero sinistro precedono la radice, che a sua volta precede tutti i nodi del sottoalbero destro
- Un albero binario è bilanciato se, per ogni nodo, i numeri dei nodi nel sottoalbero sinistro e nel sottoalbero destro differiscono al più di uno

Albero binario ordinato



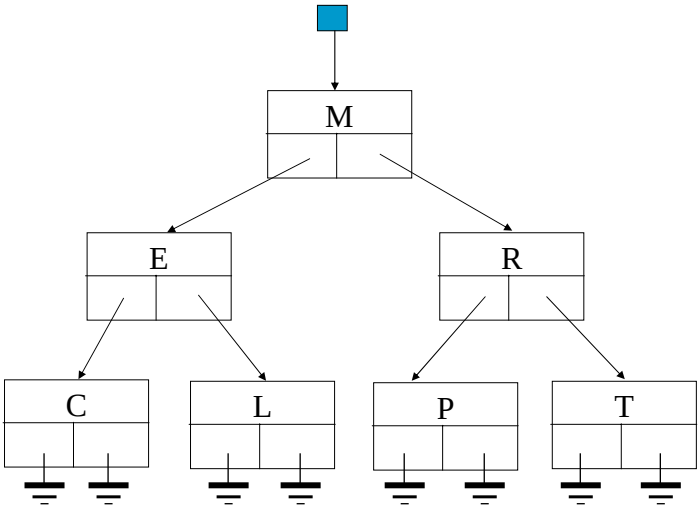
Albero binario bilanciato e non



ADT albero binario ordinato: interfaccia

ALBERO BINARIO ORDINATO			
nome	**domini**	**semantica**	**notazione**
operazioni			
Start	ϕ→A	Inizializza l'albero:produce l'albero vuoto	Start(a)
Inserisci	AxE→A	Inserisce un nodo e (in ordine)	Inserisci(a,e)
Elimina	AxE→A	Estrae un nodo e	Elimina(a,e)
Stampa	A → ϕ	Stampa le informazioni nei nodi dell'albero	Stampa(a)
predicati			
Empty	A→booleano	L'albero è vuoto	Empty(a)
Full	A→booleano	L'albero è pieno	Full(a)
Ricerca	AxE→booleano	Il nodo e è presente nell'albero	InAlb(a,e)

ADT albero binario: rappresentazione 1/2



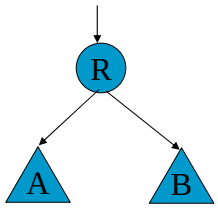
ADT albero binario: rappresentazione 2/2

- Un tipo Record adatto alla rappresentazione di un albero binario può essere il seguente:

```
struct Record* PRec;
struct Record {
  E elem;
  PRec sin;
  PRec des;
};
```

Attraversamento (visita)

- Indicate con le lettere R,A,B rispettivamente la radice, il sottoalbero sinistro e il sottoalbero destro, è possibile visitare l'albero in tre modi (ordini) diversi:
  - Preordine: R,A,B
  - Inordine: A,R,B
  - Postordine: A,B,R



Procedure di attraversamento

- Sono realizzate in forma ricorsiva

```
void preordine(const nodoptr a) {
  if (a!=0) {
     P(a);
     preordine(a->sin);
     preordine(a->des);
  }
}
```

```
void inordine(const nodoptr a) {
  if (a!=0) {
     inordine(a->sin);
     P(a);
     inordine(a->des);
  }
}
```

```
void postordine(const nodoptr a) {
  if (a!=0) {
     postordine(a->sin);
     postordine(a->des);
     P(a);
  }
}
```

P() è la procedura che svolge l'elaborazione desiderata sul nodo corrente, durante la visita.

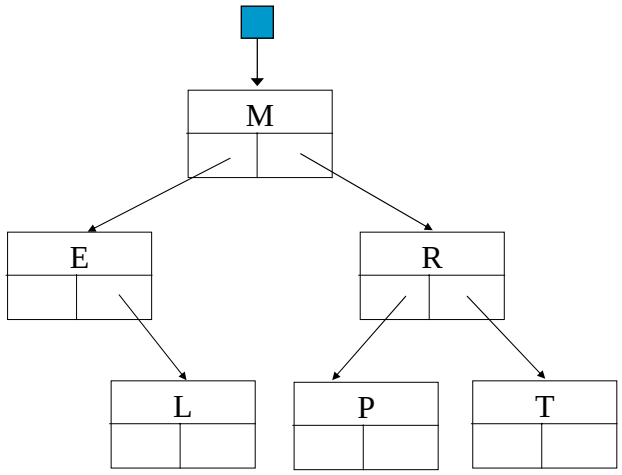
Inserimento

- Richiede la visita dell'albero per trovare il punto di inserzione
- L'inserimento in un albero ordinato può aver sempre luogo in corrispondenza di una foglia
- Lo svantaggio consiste nel fatto che non è predibile in che modo crescerà l'albero (cioè che forma assumerà)
  - non sarà necessariamente bilanciato
  - di conseguenza l'algoritmo di ricerca di un elemento richiederà un numero di confronti maggiori che nel caso di un albero bilanciato
- Esistono algoritmi di inserimento in un albero bilanciato (che mantengono "l'equilibrio")

Eliminazione

- Possono verificarsi tre casi:
  1. il nodo da rimuovere è un nodo foglia
     - il puntatore al nodo viene posto a `Null`
  2. il nodo da rimuovere ha un solo discendente
     - il discendente assume il posto del nodo da eliminare
  3. il nodo da rimuovere ha due discendenti
     - si ricopia nel nodo da eliminare il valore del nodo N_{MAX} presente nel suo sottoalbero sinistro ed avente il valore massimo
     - poiché, per costruzione, N_{MAX} può al massimo avere un solo figlio, si rimuove tale nodo dall'albero così come specificato al passo 1 o 2 (a seconda dei casi).

Esempio di eliminazione di un nodo



Esempio

- Con riferimento all'albero ordinato della figura precedente:
  - caso 1: rimozione di P
    - si pone a `Null` il puntatore a P
  - caso 2: rimozione di E
    - si sostituisce E con L
  - caso 3: rimozione di M
    - si ricerca nel sottoalbero sinistro il nodo con il valore più alto (L)
    - si scollega tale nodo dall'albero
    - si ricopia il valore di tale nodo nel nodo da eliminare (L prende il posto di M)

ADT Tabella

- Una tabella è una struttura dati definita come un insieme finito di coppie:
  - C=(x,y) $x \in X, y \in Y$ dove y=f(x)
- x è detta chiave della tabella
- y è detto valore
- I tipi X e Y possono essere semplici o strutturati

ADT Tabella: interfaccia

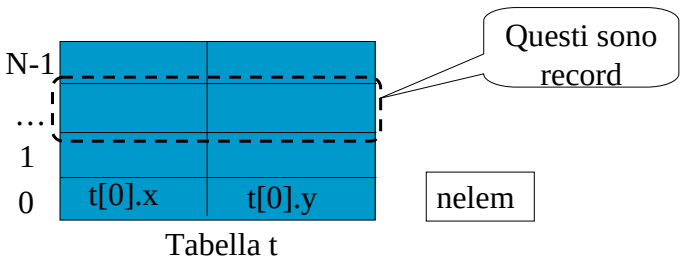
TABELLA			
nome	**domini**	**semantica**	**notazione**
operazioni			
Start	$\phi \rightarrow T$	Inizializza la tabella:produce la tabella vuota	Start(t)
Inserisci	TxC→T	Inserisce una coppia c nella tabella ($x \notin t$)	Inserisci(t,c)
Elimina	TxC→T	Se $x \in t$, elimina c da t	Elimina(t,c)
Ricerca	TxX→Cxbooleano	Cerca la chiave x, se c'è restituisce c	Cerca(t,x,c,b)
predicati			
Empty	T→booleano	La tabella è vuota	Empty(t)
Full	T→booleano	La tabella è piena	Full(t)
InTab	TxC→booleano	C è presente in t	InTab(t,c,b)

Rappresentazione

- In forma statica mediante array
- In forma dinamica mediante lista a puntatori
- Le coppie possono essere ordinate per chiave per facilitare le operazioni di accesso

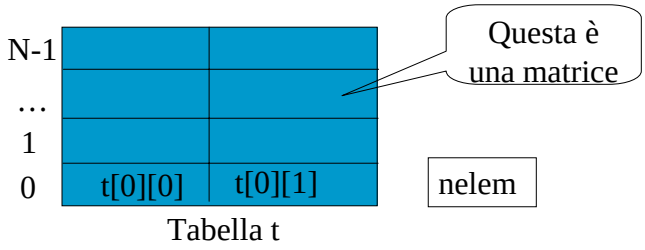
Rappresentazione statica con array di record

- Il Record rappresenta la coppia chiave-informazione
- Un unico array monodimensionale i cui elementi sono i record



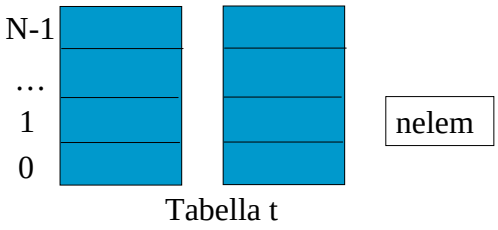
Rappresentazione statica con matrice

- Un unico array bidimensionale
- La prima colonna contiene le chiavi
- La seconda colonna i valori corrispondenti alle chiavi
- Limitazione forte: chiavi e informazioni devono essere dello stesso tipo!



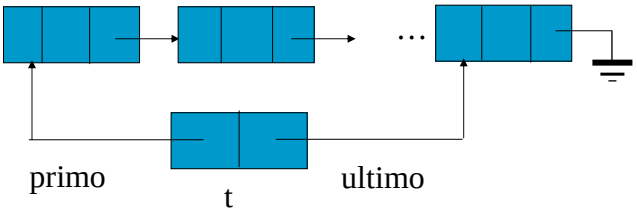
Rappresentazione statica con due array monodimensionali

- Due array monodimensionali
- Il primo contiene le chiavi
- Il secondo contiene i valori corrispondenti alle chiavi



Rappresentazione mediante lista dinamica

- Ciascun elemento della tabella è un record contenente tre campi: chiave, informazione, puntatore al prossimo.



Confronto tra le strutture concrete (1/3)

- Riassumendo il tipo di dato astratto *lista* può essere realizzato mediante le seguenti strutture concrete:
  - array non ordinato
  - array ordinato
  - lista dinamica semplicemente collegata, non ordinata
  - lista dinamica semplicemente collegata, ordinata
  - albero binario ordinato

Confronto tra le strutture concrete (2/3)

- Memoria:
  - strutture dinamiche:
    - richiedono una quantità di memoria superiore a quella strettamente necessaria per la memorizzazione dei valori
    - questa perdita è percentualmente contenuta quando i valori da memorizzare occupano molto spazio in memoria
  - array statici:
    - la perdita di memoria è in genere ancora più elevata dovendo sempre fronteggiare anche il caso peggiore

Confronto tra le strutture concrete (3/3)

- Ordinamento:
  - strutture a puntatori:
    - permettono con semplicità la realizzazione di ordinamenti multipli richiedendo una catena di puntatori per ciascun ordinamento
  - array ordinati:
    - in questo caso, poiché l'ordinamento logico è anche l'ordinamento fisico, non è possibile ordinare l'array secondo differenti criteri

Sequenze lineari: analisi comparativa 1/3

- La complessità delle operazioni elementari su una sequenza lineare (ricerca, visita, inserimento, cancellazione) varia a seconda della rappresentazione adottata. Questa va pertanto scelta considerando la frequenza relativa delle diverse operazioni
- La sequenza può essere ordinata o non ordinata. In generale:
  - in una sequenza ordinata sono più efficienti le operazioni di visita ordinata e di ricerca
  - al contrario sono meno efficienti quelle di inserimento ed eliminazione

Analisi comparativa 2/3

- Nel seguito la complessità viene valutata con riferimento al numero di:
  - confronti
  - spostamenti di un elemento della sequenza

  necessari per effettuare le suddette operazioni elementari
- Diremo:
  - n: numero di elementi nella sequenza
  - Oc(n): ordine di grandezza del numero di confronti
  - Os(n): ordine di grandezza del numero di spostamenti
  - O(n): ordine di grandezza complessivo del numero di confronti e di spostamenti

Analisi comparativa 3/3

		Array non ordinato	Array ordinato	Lista din. non ordinata	Lista din. ordinata	Albero binario
A) Ricerca	Oc(n)	n	Log n	n	n	Log n
	Os(n)	-	-	-	-	-
	O(n)	n	Log n	n	n	Log n
B) Inserimento	Oc(n)	-	Log n	-	n	Log n
	Os(n)	1	n	1	1	1
	O(n)	1	n	1	n	Log n
C) Eliminazione	Oc(n)	n	Log n	n	n	Log n
	Os(n)	n	n	1	1	1
	O(n)	n	n	n	n	Log n
D) Visita ordinata	Oc(n)	n^2	-	n^2	-	-
	Os(n)	n	n	n	n	n
	O(n)	n^2	n	n^2	n	n

N.B. la ricerca binaria è utilizzata sempre ove possibile (non sulla lista dinamica)