Enrica Reggiani

STRUTTURE DI DATI DINAMICHE

Le strutture dati sono insiemi di dati caratterizzati dalle proprietà logiche e dalla loro forma di aggregazione (vettore, matrice, pila, coda…); le possiamo analizzare da un punto di vista astratto (indipendentemente dalla memorizzazione in memoria centrale) o concreto (in relazione alla allocazione effettiva in memoria):

ASTRATTE (logiche): la struttura dati è gestita dal programmatore dal punto di vista delle operazioni di gestione (v.pila, coda) e in modo indipendente dalla memorizzazione fisica dei dati in memoria centrale.
In particolare:
PILA (STACK): è una sequenza di elementi gestiti con metodo LIFO (LAST IN FIRST OUT, l'ultimo elemento inserito è il primo ad essere estratto); le operazioni possibili sono PUSH (aggiungi,inserisci) e POP(togli,estrai): si può inserire ed estrarre solo da una parte; ad ogni push e pop la dimensione varia di una unità; quando faccio un pop (o visualizzo lo stato) devo controllare se è vuota.
CODA (QUEUE): è una sequenza di elementi gestiti con tecnica FIFO (FIRST IN FIRST OUT); anche in questo caso le operazioni possibili sono PUSH e POP: si può inserire solo da una parte, accodando l'elemento (in CODA), ed estrarre solo dalla parte opposta (TESTA); ad ogni push o pop la dimensione varia; come per la pila, devo controllare se è vuota.
Esiste anche la DOPPIA CODA: ammette la possibilità di estrarre ed inserire l'elemento in uno qualunque degli estremi
LISTA: è una successione di nodi, dove ogni nodo è in relazione con altri nodi attraverso legami detti archi; ogni nodo è costituito dai dati relativi all'elemento più un riferimento al nodo successivo. Ogni elemento conosce il suo successore e in questo modo, partendo dal primo elemento, è possibile ricostruire tutti gli elementi nella lista (che non sono necessariamente allocati in modo consecutivo).
GRAFO: è un insieme di nodi interconnessi da archi, per esprimere una relazione (ad esempio "essere amico di"); la successione dei nodi che devono essere attraversati per raggiungere un nodo prefissato è datta "cammino" (path).
ALBERO: è un grafo particolare: è un insieme di nodi connessi tra loro in modo che non esistano cicli, cioè privo di circuiti o di percorsi chiusi (un percorso chiuso permette, dato un nodo di partenza, di tornare allo stesso nodo attraverso un percorso diverso)

TDA
attributi
operazioni

In generale le STRUTTURE DATI rappresentano un esempio di tipo di dato astratto (TDA) in quanto sono composte da dati e da un insieme di operazioni che agiscono sui dati.

CONCRETE (interne): è la effettiva implementazione della struttura nel linguaggio e relativa allocazione in memoria della struttura astratta

Le strutture dati (e le tipologie di dati esistenti), inoltre, possono essere classificate anche in base alla implementazione nel linguaggio di programmazione e relativa allocazione in memoria, secondo due modalità:
STATICA: con la dichiarazione (e successiva compilazione del programma) viene assegnato in memoria centrale uno spazio fisso la cui ampiezza dipende dal tipo di variabile, dalle specifiche del compilatore e dalla macchina; infatti possiamo gestire in modo statico una pila, una coda, ecc. utilizzando un array di dimensione predefinita; in questo caso, oltre a controllare se la struttura è vuota, è necessario gestire anche il caso in cui sia piena.
DINAMICA: l'area di memoria necessaria alle variabili non viene predeterminata, ma generata in fase di esecuzione del programma. La realizzazione di strutture dinamiche comporta un'interazione continua con il GESTORE DELLA MEMORIA per allocare nuove porzioni di memoria quando la struttura di dati cresce e per deallocarle quando la struttura si contrae; in C questo compito è a carico del programmatore, attraverso la gestione dei puntatori; in Java interviene in modo automatico il GARBAGE COLLECTOR.

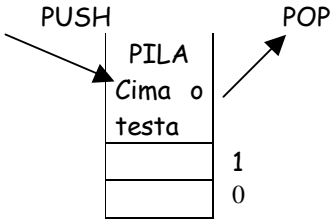
In Java, comunque, gli array non hanno dimensione prestabilita, salvo che non lo decida il programmatore; quindi l'allocazione è comunque dinamica, senza problemi di gestione della dimensione.

La classe Vector, contenuta nelle librerie standard di Java (java.util), supporta le caratteristiche di dinamicità e risulta molto utile per implementare strutture di dati dinamiche (in particolare pila e coda); questa classe gestisce un array dinamico di elementi di classe Object , la cui dimensione può aumentare e diminuire in base ai dati che contiene; l'array si adatta automaticamente: viene allocato nuovo spazio quando serve e deallocato quando non è più utilizzato (v. libro). Utilizzeremo la classe Vector per gestire la pila e la coda, i cui elementi sono allocati in modo consecutivo in memoria.

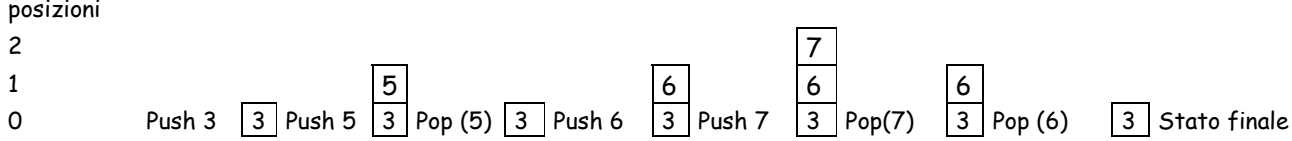
Quandi gli array, le pile e le code possono essere implementate con un linguaggio di programmazione in modo statico o in modo dinamico; finora le abbiamo implementate in modo statico, definendo a priori la dimensione massima (ed eventualmente ridimensionandole); ora li implementeremo in modo dinamico sfruttando la classe Vector (e il Garbage Collector).

PILA (stack)

È una struttura di dati dinamica gestita usando la modalità LIFO (last in-first out). Gli elementi vengono aggiunti (PUSH) ad una estremità e sono posizionati uno sopra l'altro, formando idealmente una pila di oggetti. Il prelevamento (POP) di un elemento avviene a partire da quello che si trova in cima alla pila ed è stato inserito per ultimo.



Questa struttura di dati viene usata quando si vuole tenere traccia delle informazioni per poi recuperarle procedendo in ordine inverso rispetto all'inserimento (come esempio pratico di uso di una pila si può pensare alla funzionalità offerta da molti programmi applicativi, che permette di annullare le ultime operazioni eseguite; in questo caso, tutte le operazioni eseguite dall'utente vengono memorizzate in una pila; quando viene richiesto di annullare un'operazione, il programma preleva l'informazione dalla pila e agisce di conseguenza. Il numero di operazioni che l'utente può annullare dipende dalle dimensioni riservate alla pila).
Ad esempio, date le seguenti operazioni:
push 3; push 5; pop; push 6; push 7; pop; pop
lo stato assunto assunto dalla pila dopo ogni operazione è il seguente:



La pila, che potrà contenere qualunque oggetto (Object), viene schematizzata usando i TDA nel seguente modo:

PILA
Array dinamico
costruttore Push Pop Cima Vuota Dimensione Elemento

Il costruttore istanzia una pila vuota.
Le operazioni di manipolazione sono:

- Push: inserisce un elemento in cima alla pila
- Pop: estrae un elemento eliminandolo dalla cima della pila

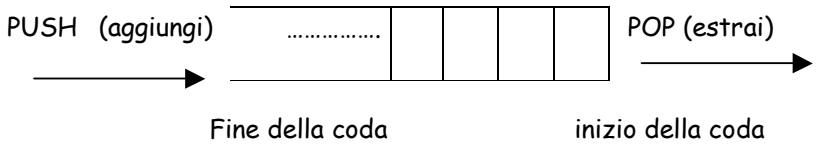
Le operazioni di interrogazione sono:

- Cima: ritorna l'elemento in cima alla pila (ad esempio per visualizzarlo)
- Vuota: segnala, con un valore booleano, se la pila è vuota
- Dimensione: ritorna la dimensione (numero di elementi) della pila
- Elemento: ritorna un elemento in una posizione specificata come parametro in ingresso

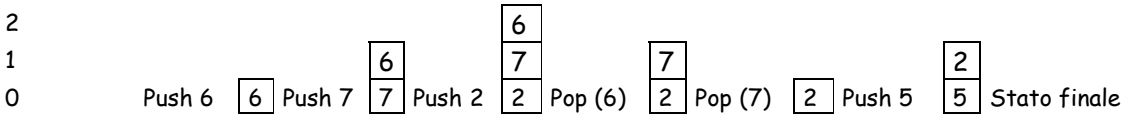
L'implementazione che faremo della PILA costituirà una classe che potrà essere riutilizzata tutte le volte che serve una struttura di dati dinamica con le caratteristiche della pila. È importante sottolineare che in questa implementazione della pila i singoli elementi sono di classe Object. Questo significa che qualunque oggetto può essere aggiunto alla pila, in quanto la classe Object è la sopraclasse di ogni altra classe. Al momento di eseguire una operazione push viene automaticamente applicato il cast verso la classe Object; quando viene prelevato un elemento, la pila lo restituisce come oggetto di classe Object: sarà compito del programmatore convertirlo tramite un ulteriore casting verso la classe originaria.
Una pila realizzata in questo modo è una struttura dati generica che può essere riutilizzata per qualunque tipo di oggetto.

CODA

E' una struttura dinamica di dati gestita usando la modalità FIFO (first in-first out). L'inserimento (PUSH) degli elementi avviene ad una estremità; essi vengono posizionati uno di seguito all'altro e formano una coda di oggetti; il prelevamento (POP) di un elemento avviene a partire da quello che è stato inserito per primo e si trova all'inizio della coda.



Questa struttura viene utilizzata quando si desidera tenere traccia delle informazioni per poi recuperarle nello stesso ordine con cui sono state inserite; la coda serve per gestire le situazioni in cui il primo che arriva è il primo ad essere servito (si pensi alla coda ad uno sportello; oppure alla gestione dello SPOOL di stampa: l'esecuzione fisica della stampa avviene nell'ordine in cui le richieste arrivano al gestore).
Ad esempio, date le seguenti operazioni:
push 6; push 7; push 2; pop; pop; push 5
lo stato assunto assunto dalla coda dopo ogni operazione è il seguente:



La coda viene schematizzata usando i TDA nel seguente modo:

CODA
Array dinamico
costruttore Push (aggiungi) Pop (estrai) vuota Dimensione Elemento

Il costruttore istanzia una coda vuota.
Le operazioni di manipolazione sono:

- Push: inserisce un elemento in fondo alla coda
- Pop: estrae un elemento all'inizio della coda

Le operazioni di interrogazione sono:

- Vuota: segnala, con un valore booleano, se la coda è vuota
- Dimensione: ritorna la dimensione (numero di elementi) della coda
- Elemento: ritorna un elemento in una posizione specifica come parametro

Come già visto per la pila, anche per la coda implementiamo in Java una classe con tali funzionalità e i cui singoli elementi sono di classe Object; questa classe potrà essere utilizzata quando occorre una struttura con le caratteristiche della coda.

LISTA CONCATENATA

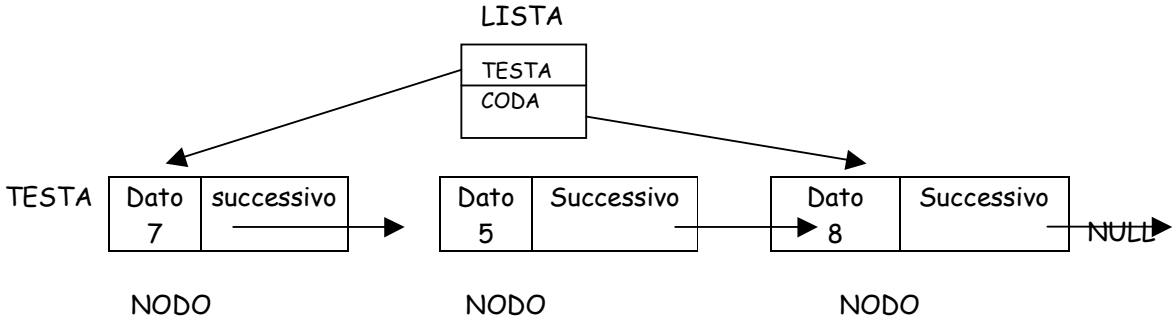
Abbiamo visto che utilizzando la classe Vector è possibile gestire in modo dinamico un array, una pila e una coda; tuttavia rimane il limite che nello heap[1] tutti i suoi elementi devono rimanere consecutivi; per risolvere questo problema si può impiegare una struttura dati in cui non è necessario che gli elementi siano memorizzati consecutivamente. La struttura idonea a questo tipo di implementazione è la LISTA CONCATENATA.

E' una struttura dati costituita da elementi (NODI) in cui ogni elemento conosce qual è e dove si trova il suo successore; in questo modo, partendo dal primo nodo, è possibile ricostruire tutti gli elementi presenti nella lista. Infatti i nodi non sono necessariamente allocati in modo consecutivo, ma ogni nodo possiede un riferimento al suo successivo

Ogni nodo quindi presenta due attributi: un attributo che contiene il dato che si vuole memorizzate, un attributo che contiene il riferimento al nodo successivo.

Per tenere traccia di questa successione è necessario conoscere il riferimento al primo elemento della lista (testa) e all'ultimo elemento (coda). Il riferimento alla testa è utile perché, partendo da questo, è possibile scorrere l'intera lista, eseguendo un'operazione chiamata attraversamento. Il riferimento alla coda serve per aggiungere rapidamente un elemento in fondo alla lista senza doverla scorrere per recuperare l'ultimo nodo. L'ultimo nodo (coda) possiede un riferimento a null, per indicare che la lista è terminata.

Graficamente una lista può essere rappresentata nel seguente modo:



In altri linguaggi di programmazione le liste vengono gestite utilizzando i puntatori; in Java invece i riferimenti agli oggetti svolgono lo stesso ruolo dei puntatori, demandando la gestione al modulo run-time di java.

La lista viene schematizzata usando i TDA.

Deve essere definito un NODO della lista nel seguente modo:

NODO
Dato Successivo
Costruttore SetSuccessivo GetSuccessivo getDato

Gli attributi sono:

- Dato: contiene l'informazione che viene memorizzata nel nodo (es. un nome, una parola….), quindi può essere String, int, … o in generale Object
- Successivo: è un riferimento al nodo successivo; l'ultimo nodo della lista assegna il valore null all'attributo Successivo; quindi è di tipo NODO (è una definizione ricorsiva)

I metodi sono:

- Costruttore: inizializza il nodo con il dato e l'attributo successivo a null
- SetSuccessivo: modifica il contenuto dell'attributo Successivo
- GetSuccessivo: ritorna il valore dell'attributo Successivo
- GetDato: ritorna il valore dell'attributo dato

Poi deve essere definita la LISTA nel seguente modo:

[1] Zona di memoria in cui vengono allocate le variabili dinamiche: array e oggetti

LISTA
Testa Coda
Costruttore Inserisci Elimina Contiene Visualizza …

Gli attributi Testa e Coda sono di tipo NODO:

- Testa: : contiene contiene il riferimento al primo elemento
- Coda: contiene il riferimento all'ultimo elemento, quindi null (consente di sapere dove termina la lista)

I metodi sono:

- Costruttore: inizializza la testa e la coda della lista a null
- Inserisci: inserisce un elemento nella lista
- Elimina: preleva un elemento specifico dalla lista
- Contiene: controlla se nella lista è presente un particolare elemento
- Visualizza: scorre la lista da testa a coda visualizzando tutti gli elementi

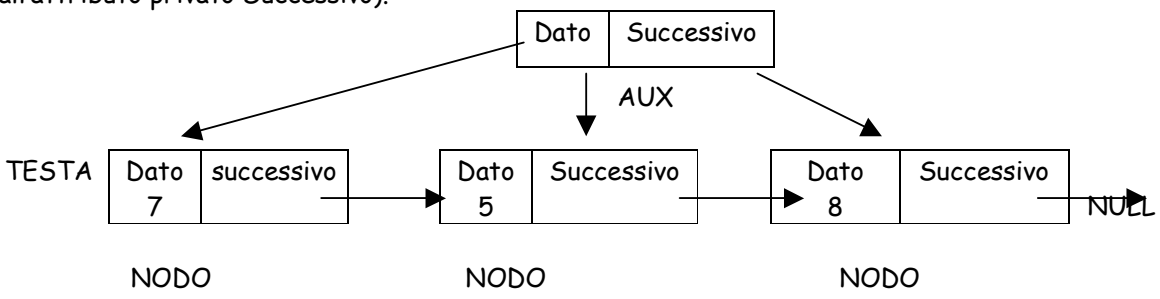
Esistono diversi tipi di liste:

- lista semplice non ordinata con legame unico (lista semplice concatenata o linkata) è il tipo più semplice, in cui ogni elemento contiene il riferimento all'elemento successivo; quindi viene percorsa a partire dal primo elemento
- lista semplice ordinata, in cui ogni nodo viene inserito rispettando un certo ordine (alfabetico, crescente…)
- lista doppia, a differenza della lista semplice, può essere percorsa nei due sensi; ogni nodo contiene due riferimenti, uno al primo (testa) e uno all'ultimo elemento (coda), che vengono sempre tenuti aggiornati.

L'operazione di inserimento può essere di vari a tipi a seconda del tipo di lista che si vuole gestire. Gli elementi possono essere aggiunti in testa, in coda oppure all'interno della lista se si è deciso di rispettare un ordine particolare nella generazione della lista; quindi ogni operazione di inserimento deve essere implementata con un metodo diverso.
Il metodo contiene è una ricerca: scorre la lista partendo dalla testa e arrestandosi quando trova l'elemento cercato oppure termina la lista (riferimento a null)
Il metodo visualizza scorre la lista dalla testa fino alla coda visualizzando gli elementi; quindi si arresta quando trova un riferimento a null.
Entrambi i metodi si basano su un ciclo di questo tipo:

```
nodo AUX=testa;
while(AUX!=null)
    {……
     //operazioni da svolgere
      AUX=AUX.getSuccessivo() ;
     }
```

viene dichiarata una variabile ausiliaria AUX di tipo nodo e inizializzata a testa : quindi contiene un riferimento alla testa della lista; mentre il riferimento di questa variabile non è null (cioè la lista non è terminata) vengono svolte le operazioni necessarie, tra cui l'istruzione AUX=AUX.getSuccessivo() che assegna ad AUX il riferimento al nodo successivo (utilizzando il metodo di get che permette di accedere all'attributo privato Successivo).

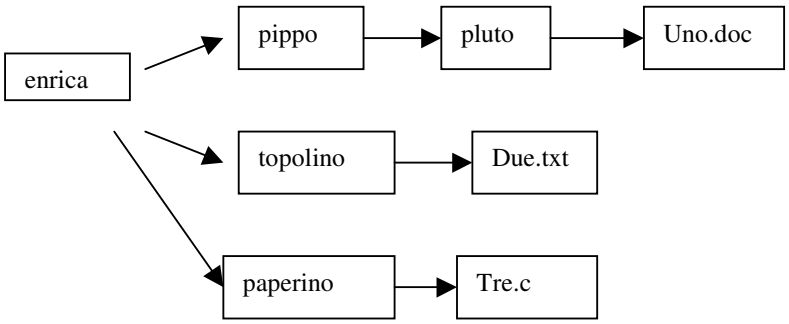


ALBERO

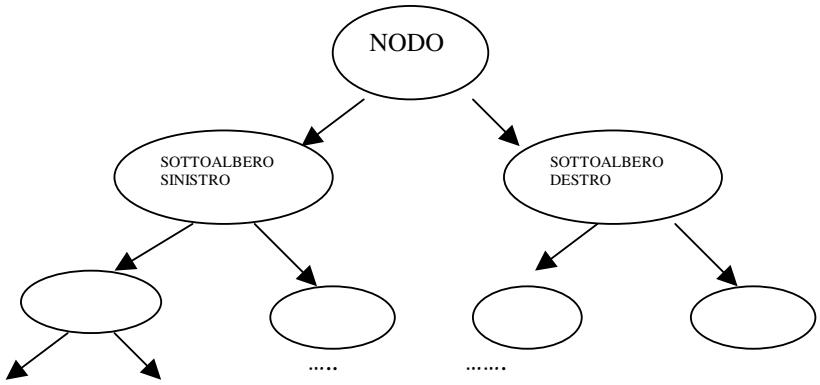
L'albero è una struttura di dati dinamica costituita da un insieme di NODI connessi tra loro in modo che non ci siano percorsi chiusi; ciò significa che, partendo da un nodo, per tornare allo stesso nodo deve essere percorso lo stesso cammino (path).
Un RAMO indica un cammino composto da più nodi connessi; i nodi che compongono l'albero sono:

- la RADICE (root): è il nodo da cui partono le connessioni che generano l'albero, il nodo da cui partono i rami
- le FOGLIE: sono i nodi esterni che non hanno figli, cioè nodi al termine di un ramo
- i NODI INTERNI: sono nodi che non sono né radice, ne foglie

il fatto che l'albero non contenga circuiti garantisce che, partendo dalla root, si possa raggiungere qualsiasi foglia senza finire in un circolo da cui non si può uscire.
un esempio di albero è l'organizzazione dei file gestita dal file system del sistema operativo; la radice è la directory principale, i nodi interni sono le altre directory, le foglie sono i file



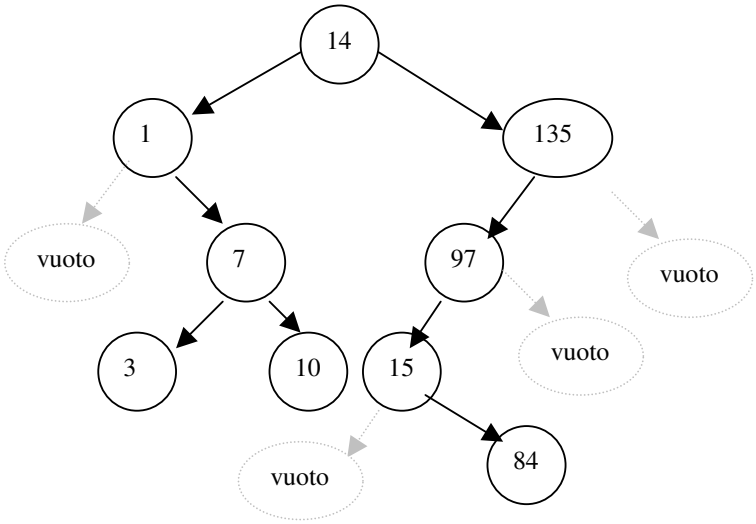
Enrica è la root, uno.doc, due.txt, tre.c sono foglie, pippo, pluto… sono nodi interni.
Un albero particolare è l'ALBERO BINARIO, nel quale ogni nodo ha al massimo due nodi figli chiamati SOTTOALBERO SINISTRO e SOTTOALBERO DESTRO; i due sottoalberi sono, a loro volta, alberi binari e possono essere vuoti oppure possono essere costituiti da nodi con due sottoalberi; cioè, un albero binario è un insieme vuoto oppure è costituito da un nodo a cui sono collegati al massimo due alberi binari:



L'albero binario è una struttura adatta per la gestione delle liste ordinate; i dati vengono immessi nell'albero secondo questo criterio: i dati che precedono il dato del nodo vanno nel sottoalbero sinistro, quelli che seguono il dato del nodo, nel sottoalbero di destra.
Ad esempio, data la lista non ordinata di numeri:

14 135 97 1 7 3 15 84 10

si vuole costruire l'albero binario per mantenere la lista dei numeri ordinata in modo crescente:



Il primo numero (14) è la radice dell'albero e separa i due numeri successivi in due sottoalberi: i numeri minori di 14 vanno nel sottoalbero di sinistra, quelli maggiori di 14 nel sottoalbero di destra.
Quindi 135 va a destra, diventa un nodo a destra della radice e dividerà i numeri maggiori di 14 in due sottoalberi: i numeri minori di 135 (ma maggiori di 14) andranno nel sottoalbero di sinistra, quelli maggiori di 135 nel destro.
Il 97, essendo maggiore di 14, va nel sottoalbero destro, e qui, essendo minore di 135, nel sottoallbero sinistro.
Il numero 1, minore di 14, va a sinistra e dividerà i numeri minori di 14 in due sottoalberi: i numeri minori 1 (qui non ce ne sono) nel sottoalbero sinistro, i numeri maggiori di 1 nel destro.
Il numero 7, minore di 14, va a sinistra e qui, essendo maggiore di 1, va a destra; il numero 3, minore di 14, va a sinistra, ma essendo maggiore di 1, va a destra e quindi, essendo minore di 7, va a sinistra; il numero 15 è maggiore di 14 (va a destra), minore di 135 (va a sinistra), minore di 97 (va a sinistra); il numero 84 va a destra di 14, a sinistra di 135, a sinistra di 97 ma a destra di 15. Infine il numero 10 va a sinistra di 14, a destra di 1, a destra di 7.
Quindi, partendo dalla radice, individuiamo diversi alberi e sottolaberi: l'albero con radice 14, sottoalbero sinistro 1 e destro 135; a sua volta il sottoalbero sinistro contiene il sottoalbero con radice 1, sottolabero sinistro vuoto e sottoalbero destro con 7, e così via.

Un albero può essere attraversato in tre modi, a seconda di come si volgiono elaborare i dati contenuti: in INORDINE (o simmetrico), in PREORDINE (o ordine anticipato), in POSTORDINE (o ordine posticipato)

INORDINE
Sottoalbero sinistro
Nodo
Sottoalbero destro

PREORDINE
Nodo
Sottoalbero sinistro
Sottoalbero destro

POSTORDINE
Sottoalbero sinistro
Sottoalbero destro
Nodo

L'attraversamento di ogni sottoalbero avviene con le stesse modalità dell'attraversamento dell'albero: se l'attraversamento dell'albero è posticipato anche l'attraversamento di ciascun sottoalbero deve essere posticipato.
Dato l'albero precedente, l'attraversamento in inordine elabora i dati nell'ordine: **1,3,7,10,14,15,84,97,135**
Infatti si parte dall'albero 1,14,135 e si percorre a sinistra: qui si considera il sottoalbero vuoto, 1, 7: essendo il sottoalbero sinistro vuoto, si prende il nodo **1**; poi si va a destra e si considera il sottoalbero

3,7,10 e si percorre a sinistra: **3**; non ci sono altri sottoalberi a sinistra, quindi si accede ai nodi: prima il nodo del sottoalbero 3,7,10: **7**; poi si va a destra: **10;** avendo percorso tutta la parte sinistra, si accede al nodo (root): **14**; ora si percorre l'albero destro, ma accedendo subito al sottoalbero sinistro (vuoto, 15, 84) e si considera subito il nodo **15** (essendo vuoto a sinistra) e poi a destra **84**; poi il sottoalbero 15,97,vuoto e si prende il nodo **97** (15 a sinistra è già stato percorso e a destra è vuoto); infine il sottoalbero97,135,vuoto): sinistra (97) è già stato percorso, quindi si considera il nodo **135** (a destra è vuoto).
L'attraversamento in <u>preordine</u> è: 14,1,7,3,10,135,97,15,84
Si parte dall'albero 1,14,135 e si prende il nodo **14**; si va a sinistra e si considera il sottoalbero vuoto,1,7; si prende il nodo **1**; a sinistra è vuoto e si va a destra; si considera il sottoalbero 3,7,10: si prende il nodo **7**, poi a sinistra **3** e a destra **10**; ora si ricomincia dell'albero destro: considero il sottoalbero 97,135, vuoto e si prende il nodo **135**, poi a sinistra considero l'albero 15,97,vuoto e si prende il nodo **97**; poi a sinistra il **15** e a destra **84**
L'attraversamento in <u>postordine</u> è: 3,10,7,1,84,15,97,135,14, leggendo prima il sottoalbero sinistro, il sottoalbero destro e poi il nodo.

Quindi, se i dati di un albero binario sono stringhe (o interi…), per visualizzarle in ordine alfabetico (o crescente…), si deve percorrere l'albero in inordine (che è il criterio con cui i dati sono stati immessi).

In un albero il numero massimo di nodi toccati per arrivare a una foglia è detto PROFONDITA' (o LIVELLI): nel caso precedente la profondità era 5. Il numero 14 è al livello 1, i numeri 1 e 135 al livello 2 e così via.
La tabella evidenzia quanti dati possono essere contenuti in albero al variare della profondità e quanti dati si possono trovare al livello più profondo (foglie):

profondità	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Numero max nodi	1	3	7	15	31	63	127	255	511	1023
Numero max foglie	1	2	4	8	16	32	64	128	256	512

Se P=profondità, N= numero massimo nodi, F=numero massimo foglie, si calcola:
$N=2^P-1$ oppure
$F=2^{(P-1)}$
Anche l'albero binario può essere schematizzato con i TDA.
Deve essere definito un NODO dell'ALBERO nel seguente modo:

NODO
Dato Eliminato Sinistro Destro
costruttore metodi di get metodi di set stampa

Gli attributi sono:

- Dato: contiene l'informazione che viene memorizzata nel nodo (es. un nome, una parola….), quindi può essere String, int, …
- Eliminato: è un attributo booleano che specifica se il nodo è stato eliminato
- Sinistro e destro sono di tipo NODO e specificano i riferimenti ai nodi che identificano il sottoalbero sinistro e destro

I metodi:

- Costruttore: istanzia un nodo con Dato, eliminato a false, sinistro e destro a null
- Metodi di get che ritornano il valore degli attributi
- Metodi di set, per modificare gli attributi eliminato, sinistro e destro
- stampa: visualizza il contenuto di dato oppure specifica se il nodo è stato eliminato

Poi deve essere definito L'ALBERO nel seguente modo:

ALBERO
Radice
Costruttore Get radice Inserisci Elimina Contiene Stampa inordine Stampa preordine Stampa postordine

Ha un solo attributo Radice di tipo NODO

I metodi sono:

- costruttore: istanzia un nodo radice a null
- get radice:ritorna il contenuto della radice
- Inserisci: inserisce un elemento nell'albero
- Elimina: elimina un elemento dall'albero
- Contiene: controlla se nell'albero è presente un particolare elemento
- Stampa inordine, preordine, postordine, visualizza il contenuto dell'albero in base al criterio di attraversamento stabilito