SPECIFICHE

1. LISTE

- Una lista è una sequenza finita, anche vuota, di elementi dello stesso tipo.
- Gli elementi possono comparire anche più volte, a differenza dell'insieme.
- Gli elementi vengono definiti atomi o nodi.
- E' possibile accedere solo al primo elemento della sequenza, per accedere agli altri bisogna scandire gli elementi che lo precedono.
- La lista è una struttura dati dinamica, è possibile variarne dinamicamente la dimensione.

1.1 Specifica sintattica

Tipi: lista, posizione, boolean, tipoelem

Operatori:
- crealista : () → lista
- listavuota : (lista) → boolean
- leggilista : (posizione, lista) → tipoelem
- scrivilista : (tipoelem,posizione,lista) → lista
- primolista : (lista) → posizione
- finelista : (posizione, lista) → boolean
- succlista : (posizione, lista) → posizione
- predlista : (posizione, lista) → posizione
- inslista (tipoelem,posizione,lista) → lista
- canclista : (posizione, lista) → lista

1.2 Specifica semantica

Tipi:
- lista: insieme delle sequenze l = <a1,…,an> con n >= 0 di elementi di tipo tipo elem dove l'i-esimo elemento ha valore ai e posizione pos(i)
- boolean: insieme dei valori verità

Operatori:
- creaLista() = l'
  POST: l' = <>
- listaVuota(l) = b
  POST: b = VERO se l = <>, altrimenti FALSO
- leggiLista(p,l) = a
  PRE: p = pos(i) con 1 <= i <= n
  POST: a = a(i) (o a = l(i) ?)

-scriviLista(a,p,l) = l'
  PRE: p = pos(i) con 1 <= i <= n
  POST: l' = <a1,…,ai-1,a,ai+1,…,an>

-primoLista(l) = p
  POST: p = pos(1)

-fineLista(p,l) = b
  PRE: p = pos(i) con 1 <= i <= n+1
  POST: b = VERO se p = pos(n+1), FALSO altrimenti

-succLista(p,l) = p'
  PRE: p = pos(i) con 1<= i <= n
  POST: p' = pos(i+1)

-precLista(p,l) = p'
PRE: p = pos(i) con 1 <= i <= n
POST: p' = pos(i-1)
-insLista(a,p,l) = l'
PRE: p = pos(i) con 1<= i <= n+1
POST: se 1<= i <= n allora l' = <a1,...,ai-1,a,ai,ai+1,...,an>
se i = n+1 allora l' = <a1,...,an,a>
se i = 1 e l = <> allora l' = <a>

-cancLista(p,l)=l'
PRE: p = pos(i) con 1 <= i <= n
POST: l' = <a1,...,a-1,a+1,...,an>

1.3 Realizzazione sequenziale con vettore
Una lista può essere rappresentata da un vettore, poiché il numero di elementi che compongono la lista può variare, si utilizza una variabile PRIMO per indicare dov'e' memorizzato il primo elemento della lista. Si usa un altra variabile LUNGHEZZA per indicare il numero di elementi da cui è composta.
La banale inserzione o cancellazione di un elemento nella lista pero' causa lo spostamento degli elementi in tutto il vettore.

1.4 Rappresentazione collegata
L'idea è quella di memorizzare gli elementi della lista associando un riferimento che individua la locazione dell'elemento successivo.

1.5 Rappresentazione collegata con cursori
Si utilizza sempre un vettore ma il problema dell'inserimento e cancellazione degli elementi viene superato attraverso i riferimenti. I riferimenti vengono realizzati tramite cursori, cioe' variabili il cui valore è interpretato come indice di un vettore.
Il vettore contiene tutte le liste, ognuna individuata da un proprio cursore iniziale. Tutte le celle libere vengono organizzate in una lista, detta “listalibera”. E' necessario che “listalibera” venga inizializzata all'inizio per collegare tra loro tutte le celle libere del vettore spazio.
Rimane comunque il problema connesso agli array di definire una dimensione.
Gli aggiornamento comunque richiedono un numero limitato di operazioni.
Rispetto alla rapp. Sequenziale vi è un ulteriore occupazione di memoria per i riferimenti.

1.6 Rappresentazione collegata con puntatori
Si fa un uso congiunto di puntatori e record, utilizziamo i puntatori per indicare l'indirizzo di locazione di memoria dell'elemento successivo. Le operazioni disponibili sul puntatore sono l'accesso alla locazione il cui indirizzo è memorizzato nel puntatore, la richiesta di una nuova locazione di memoria e il rilascio della locazione di memoria.

2. PILE

Una pila è una sequenza di elementi di un certo tipo in cui è possibile aggiungere o togliere elementi solo daun estremità (testa). Non è possibile accedere a qualsiasi altro elemento che non sia quello in testa

2.1 Specifica sintattica

Tipi: pila, boolean, tipoelem

Operatori:
- creaPila: () → pila
- pilaVuota: (pila) → boolean
- leggiPila: (pila) → tipoelem
- fuoriPila: (pila) → pila
- inPila: (tipoelem, pila) → pila

2.2 Specifica semantica

Tipi:
- pila: sequenza di elementi p = <a1,…,an> di tipo tipoelem gestiti con accesso LIFO
- boolean: insieme dei valori di verità

Operatori:
- creaPila() = p
  POST: p = <>
- pilaVuota(p) = b
  POST: b = VERO se p = <>, altrimenti FALSO
-leggiPila(p) = a
  PRE: p = <a1,…,an> con n >= 1
  POST: a = a1
-fuoriPila(p) = p'
  PRE: p = <a1,…,an> con n >= 1
  POST : p' = <a2,…,an> se n >1, se n = 1 allora p' = <>
-inPila(a,p) = p'
  PRE: p = <a1,…,an> con n >= 0
  POST: se n > 0 allora p' = <a,a1,…,an>, se n = 0 allora p' = <a>

2.3 Realizzazione

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

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)

2.4 Realizzazione con vettore

Con questa realizzazione ogni operatore richiede tempo costante per essere eseguito. Questa realizzazione presenta due svantaggi: richiede di determinare a priori un limite al numero max di elementi della pila in quanto bisogna stabilire la dimensione del vettore a priori. Gli elementi vengono memorizzati in ordine inverso mantenendo un cursore alla testa

della pila, cioè l'ultimo elemento inserito nel vettore, in fondo. Quindi non si richiedono spostamenti per cancellazioni e aggiunte.

2.5 Realizzazione con puntatori

Come con le liste, utilizziamo un puntatore alla prima cella che sarebbe la testa della pila.

3. CODE

Una coda è un tipo astratto simile alla lista che permette di inserire gli elementi in coda e recuperarli dalla testa (FIFO), non è possibile accedere agli altri elementi

3.1 Specifica sintattica

Tipi: coda, tipoelem, boolean
Operatori:
- creaCoda: () → coda
- codaVuota: (coda) → boolean
- inCoda: (tipoelem, coda) → coda
- fuoriCoda: (coda) → coda
- leggiCoda: (coda) → tipoelem

3.2 Specifica semantica

Tipi:
coda: insieme delle sequenze di elementi P = <a1,…,an> con n >= 0 di tipo tipoelem i cui accessi sono gestiti in FIFO
boolean: insieme dei valori di verità

Operatori:
- creaCoda() = p
  POST: p = <>
- codaVuota(p) = b
  POST: b = VERO se p = <>, FALSO altrimenti
- inCoda(a,p) = p'
  PRE: p = <a1,…,an> con n >= 0
  POST: se n > 0 allora p' = <a1,…,an,a>, se n = 0 allora p' = <a>
-fuoriCoda(p) = p'
  PRE: p = <a1,…,an> con n >= 1
  POST: se n > 1 allora p' = <a2,…,an>, se n = 1 allora p' = <>
-leggiCoda(p) = a
  PRE: p = <a1,…,an> con n >= 1
  POST: a = a1

3.3 Realizzazione

In generale la rappresentazione di una coda è analoga a quella di una pila con l'attenzione di garantire l'accesso sia al primo elemento inserito che all'ultimo.

3.4 Realizzazione con vettore circolare

La rappresentazione di una coda con vettore non è agevole come per le pile, è utile gestire l'array come circolare: Il vettore è inteso di dimensioni maxlung e l'elemento maxlung punterà come successivo al primo elemento del vettore. In questo modo possiamo continuare a cancellare dalla testa e inserire in coda senza dover operare spostamenti tra gli elementi restanti.

3.5 Realizzazione con puntatori

Garantire l'accesso al primo elemento e ultimo elemento inserito.

4. INSIEMI

Un insieme è una collezione di elementi di tipo omogeneo. A differenza delle liste gli elementi non sono caratterizzati da una posizione, ne possono apparire più di una volta. La relazione fondamentale è quella di appartenenza.

4.1 Specifica sintattica

Tipi: insieme, tipoelem, boolean

Operatori:

- creaInsieme:	()	→	insieme
-insiemeVuoto:	(insieme)	→	boolean
-appartiene:	(tipoelem,insieme)	→	boolean
- inserisci:	(tipoelem,insieme)	→	insieme
- cancella:	(tipoelem,insieme)	→	insieme
- unione:	(insieme, insieme)	→	insieme
- intersezione:	(insieme, insieme)	→	insieme
- differenza:	(insieme, insieme)	→	insieme

4.2 Specifica semantica

Tipi:
- insieme: famiglia di insiemi di elementi di tipo tipoelem
- booelan: insieme dei valori di verità

Operatori:
- creaInsieme() = P
  POST: P = {}
-insiemeVuoto(p) = b
  POST: b = VERO se P = {}, altrimenti FALSO
-appartiene(a,P) = b
  POST: b = VERO se a ∈ P, b = FALSO altrimenti
- inserisci(a,P) = P'
  PRE: a ∉ P
  POST: P' = P U {a}, se a ∈ P allora P' = P
- cancella(a,P) = P'
  PRE: a ∈ P
  POST: P' = P / {a}, se a ∉ P allora P' = P
-unione(A,B) = P
  POST: P = A U B
- intersezione(A,B) = P
  POST: P = A ^ B
- differenza(A,B) = P
  POST: P = A / B

4.3 Rappresentazione con vettore booleano

E' possibile rappresentare un insieme A i cui elementi sono ad esempio interi in [1,n] attraverso un vettore booleano di bit, il cui k-esimo vaolore sarà VERO se k ∈ A, FALSO altrimenti (vettore caratteristico).

4.4 Rappresentazione con liste non ordinate

L'occupazione di memoria è proporzionale al numero di elementi presenti nell'insieme,

l’inserimento avviene in testa alla lista semplice con cui è realizzato l’insieme.

4.5 Rappresentazione con liste ordinate

Se è definita una relazione <= di ordinamento totale sugli elementi dell’insieme, esso può essere rappresentato con una lista ordinata per valori crescenti degli elementi, utilizzando due puntatori che scorrono ognuno su un insieme.

5. DIZIONARI

I dizionari sono sottotipi di insieme per i quali non si richiedono tutte le operazioni definite sull'insieme. Gli elementi sono generalmente tipi strutturati ai quali si accede per mezzo di un riferimento ad un campo chiave. Gli elementi assumono la forma di una coppia costituita da (chiave, valore). La caratteristica della chiave è legata all'applicazione, il valore associato invece rappresenta l'informazione associata per scopi di gestione.

Le operazioni su un dizionario devono consentire la verifica dell'esistenza di una definita chiave e deve essere possibile l'inserimento di una nuova coppia come pure la cancellazione. In più è necessario fornire operazioni per il recupero delle informazioni valore e la loro modifica.

5.1 Specifica sintattica

Tipi: dizionario, chiave, valore, boolean
Operatori:

- creaDizionario: () → dizionario
- dizionarioVuoto (dizionario) → boolean
- appartiene (chiave,dizionario) → boolean
- inserisci (<chiave,valore>,dizionario) → dizionario
- cancella (chiave, dizionario) → dizionario
- recupera (chiave,dizionario) → valore

5.2 Specifica semantica

Tipi:
dizionario: famiglia degli insiemi(dizionari) di coppie di tipo <chiave, valore>
boolean: insieme dei valori di verità

Operatori:

- creaDizionario = D
  POST: D = {}
- dizionarioVuoto(D) = b
  POST: b = VERO se D = {}, altrimenti FALSO
- appartiene(k,D) = b
  POST: b = VERO se esiste una coppia <k',v'> ∈ D per cui k' = k, FALSO altrimenti
- inserisci(<k,v>,D) = D'
  POST: D' = D U {<k,v>} se non esista una coppia <k',v'> tale che k' = k
  D' = D U {<k,v>} / {<k',v'>} se esiste gia' una coppia <k',v'> tale che k' = k
- cancella(k,D) = D'
  PRE: esiste una coppia <k',v'> ∈ D tale che k' = k
  POST: D' = D / {<k',v'>} tale che k = k'
- recupera(k,D) = v
  PRE: esiste una coppia <k',v'> ∈ D tale che k' = k
  POST: v = v'

5.3 Rappresentazioni

Oltre alle realizzazioni viste per l'insieme che si rifanno alle rappresentazioni con vettore booleano e mediante lista, ci sono realizzazioni più efficienti mediante vettori ordinati e tabelle hash.

5.4 Rappresentazione con vettore ordinato
SI utilizza un vettore con un cursore all'ultima posizione occupata. Avendo definito una relazione di ordine totale <= sulle chiavi queste si memorizzano nel vettore in posizioni contigue e in ordine crescente a partire dalla prima posizione. Per verificare l'appartenenza di un elemento si utilizza la ricerca binaria.

5.5 Rappresentazione con tabella hash
Esiste una tecnica denominata hashing che si appoggia su una struttura dati tabellare che si presta ad essere usata per realizzare dizionari. Con questa struttura le operazioni di ricerca e modifica di un dizionario possono operare in tempi costanti e indipendenti sia dalla dimensione del dizionario che dall'insieme dei valori che verranno gestiti.
- Idea base: ricavare la posizione che la chiave occupa in un vettore dal valore della chiave.
- Esistono diverse varianti che comunque si possono far risalire ad una forma statica e ad una forma dinamica o estensibile.

L'hash statico fa uso di strutture di dimensioni prefissate mentre l'hash dinamico è in grado di modificare dinamicamente le dimensioni della tabella hash sulla base del numero di elementi vengono inseriti o eliminati.

L'hash statico può assumere a sua volta due forme diverse:
- Hash chiuso: consente di inserire un insieme limitato di valori in uno spazio a dimensione fissa.
- Hash aperto: consente di memorizzare un insieme di valori di dimensione qualsiasi in uno spazio potenzialmente illimitato

Entrambe queste varianti però utilizzano una sottostante tabella hash a dimensione fissa .

- Nel caso di hash chiuso la struttura sarà composta da un certo numero (maxbucket) di contenitori di uguale dimensione denominati bucket. Ognuno di questi bucket può contenere al massimo un numero nb = 1 di elementi che comprendono la chiave e il corrispondente valore.
- Nel caso di hash aperto la struttura sarà composta da un certo numero indeterminato di contenitori bucket.

In entrambi i casi viene usata una funzione aritmetica allo scopo di calcolare partendo dalla chiave, la posizione in tabella delle informazioni contenute nell'attributo collegato alla chiave.

5.6 Collisioni
Una collisione si verifica quando chiavi diverse producono lo stesso risultato della funzione hash. Esistono funzioni hash più o meno buone anche se le collisioni non si potranno mai evitare del tutto.

5.6.1 Gestione con scansione lineare
Se h(k) per qualche chiave k indica una posizione già occupata, si ispeziona lea posizione successiva nel vettore. Se la posizione è piena, si prova con la seguente fino a trovare una posizione libera o trovare che la tabella è completamente piena. Una posizione libera può essere facilmente segnalata in fase di realizzazioneda una chiave fittizia "libera". Per la cancellazione basta sostituire la chiave con una fittizia "cancellato". La strategia lineare può produrre nel tempo il casuale addensamento di informazioni in certi tratti della tabella (agglomerati) piuttosto che una loro dispersione.

Quale sia la funzione hash adottata, deve essere prevista una strategia per gestire il problema degli agglomerati e delle collisioni. Cioè:
- occorre una funzione hash che distribuisca le chiavi uniformemente in v, in modo da ridurre le collisioni.
- occorre un metodo di scansione per la soluzione delle collisioni utile a reperire chiavi che hanno trovato la posizione occupata e che non provochi la formazione di agglomerati di chiavi.
- la dimensione m del vettore v deve essere una sovrastima del numero delle chiavi attese per evitare di riempire v completamente.

5.6.2 Hash aperto e liste di trabocco

Una tecnica che evita la formazione di agglomerati è quella dell'hash aperto che richiede che la tabella hash mantenga la lista degli elementi le cui chiavi producono lo stesso valoere di funzione. La tabella viene realizzata definendo un array di liste di bucket (liste di trabocco). La funzione hash viene utilizzata per definire quale lista potrebbe contenere l'elemento che possiede una determinata chiave in modo da poter attivare una successiva operazione di ricerca nella lista corrispondente e di restituire la posizione del bucket contenente la chiave.

Nella realizzazione con hash e liste di trabocco si usa un metodo di scansione esterno contrapposto alla scansione lineare che è un metodo interno, altri metodi interni sono la scansione quadratica, pseudocasuale e l'hashing doppio.

6. ALBERI BINARI

L'albero è una struttura informativa fondamentale per rappresentare partizioni successive di un insieme in sottoinsiemi disgiunti, organizzazioni gerarchiche di dati, procedimenti decisionali enumerativi. L'albero è un particolare tipo di grafo, definto come una coppia T = (N, A) dove N è un insieme finito di nodi ed A è un insieme di coppie non ordinate tale che:

- Il numero di archi è uguale a |A| = |N| - 1
- T è connesso, ovvero per ogni coppia di nodi u, v esiste una sequenza di nodi distinti u0, …,uk tali che u = u0, v = uk e la coppia <ui,ui+1> è un arco di A per i = 0,…,k-1

Gli alberi binari sono particolari alberi ordinati in cui ogni nodo ha al più due figli e si fa sempre distinzione tra figlio sinistro, che viene prima nell'ordinamento, e figlio destro. Un albero binario è un grafo orientato che o è vuoto, o è costituito da un solo nodo, o è formato da un nodo N detto radice e da due sottoalberi binari che vengono chiamati sottoalbero sinistro e destro.

6.1 Specifica sintattica

Tipi: alberoBin, nodo, boolean, tipoElem

Operatori:

creaAlberoBin:	()	→	alberoBin
alberoBinVuoto:	(alberoBin)	→	boolean
binRadice:	(alberoBin)	→	nodo
binPadre:	(nodo, alberoBin)	→	nodo
figlioSinistro:	(nodo, alberoBin)	→	nodo
figlioDestro:	(nodo, alberoBin)	→	nodo
sinistroVuoto:	(nodo,alberoBin)	→	boolean
destroVuoto:	(nodo, alberoBin)	→	boolean
leggiNodo:	(nodo,alberoBin)	→	tipoElem
scriviNodo:	(tipoElem, nodo, alberoBin)	→	alberoBin

- Operatori per la costruzione dell'albero dalle foglie fino alla radice:

costrAlberoBin	(alberoBin, alberoBin)	→	alberoBin
cancSottoAlberoBin	(nodo, alberoBin)	→	alberoBin

- Operatori per la costruzione dell'albero dalla radice fino alle foglie:

insBinRadice:	(nodo, alberoBin)	→	alberoBin
insFiglioSinistro:	(nodo, alberoBin)	→	alberoBin
insFiglioDestro:	(nodo, alberoBin)	→	alberoBin

6.2 Specifica semantica

Tipi:

- alberoBin: insieme degli alberi binari T=(N, A) nei quali ad ogni nodo è associato un Livello, boolean, nodo.

Operatori:

- creaAlberoBin() = T
  POST: T = (∅, ∅) = Λ
- alberoBinVuoto(T) = b
  POST: b = VERO se T = Λ, altrimenti FALSO

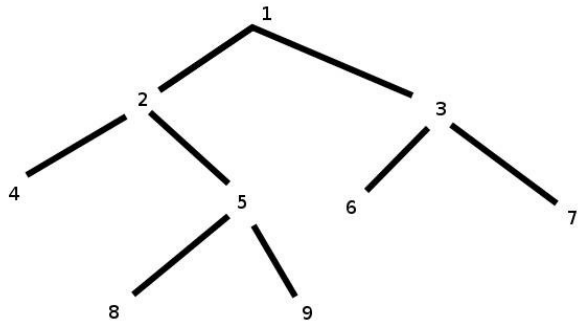
- binRadice(T) = u
  PRE: $T \neq \Lambda$
  POST: u è radice di T e livello(u) = 0
- binPadre(u, T) = v
  PRE: $T \neq \Lambda$, $u \in N$, livello(u) > 0
  POST: v è padre di u, livello(v) = livello(u) − 1
- figlioSinistro(u, T) = v
  PRE: $T \neq \Lambda$, $u \in N$, u ha un figlio sinistro
  POST: v è figlio sinistro di u in T, livello(v) = livello(u) + 1
- figlioDestro(u, T) = v
  PRE: $T \neq \Lambda$, $u \in N$, u ha un figlio destro
  POST: v è figlio destro di u in T, livello(v) = livello(u) + 1
- sinistroVuoto(u, T) = b
  PRE: $T \neq \Lambda$, $u \in N$
  POST: b = VERO se u non ha figlio sinistro, altrimenti FALSO
- destroVuoto(u, T) = b
  PRE: $T \neq \Lambda$, $u \in N$
  POST: b = VERO se u non ha figlio destro, altrimenti FALSO
- costrBinAlbero(X, Y) = T
  POST: T si ottiene da X e Y introducendo un nuovo nodo r (radice di T) che avrà come sottoalbero sinistro X e sottoalbero destro Y.
  - Se $X = \Lambda$ e $Y = \Lambda$ allora la radice r non avrà figli.
  - Se $X = \Lambda$ allora la radice r non avrà figlio sinistro.
  - Se $Y = \Lambda$ allora la radice r non avrà figlio destro.
- cancSottoAlberoBin(u, T) = T'
  PRE: $T \neq \Lambda$, $u \in N$
  POST: T' è ottenuto da T eliminando il sottoalbero di radice u con tutti i suoi discendenti. Valido per alberi di ogni ordine.
- leggiNodo(u, T) = a
  PRE: u è nodo di T, $u \in N$
  POST: a è l'etichetta associata al nodo u in T
- scriviNodo(a, u, T) = T'
  PRE: u è un nodo di T, $u \in N$
  POST: T' è il nuovo albero corrispondente al vecchio T con il valore a associato al nodo u.
- insBinRadice(u, T) = T'
  PRE: $T = \Lambda$
  POST: $T' = (N, A)$, $N = \{u\}$, livello(u) = 0, $A = \varnothing$
- insFiglioSinistro(u, T) = T'
  PRE: $T \neq \Lambda$, $u \in N$, sinistroVuoto(u, T) = VERO
  POST: $N' = N \cup \{v\}$, T' è ottenuto da T aggiungendo v come figlio sinistro di u
- insFiglioDestro(u, T) = T'
  PRE: $T \neq \Lambda$, $u \in N$, destroVuoto(u, T) = VERO
  POST: $N' = N \cup \{v\}$, T' è ottenuto da T aggiungendo v come figlio destro di u

6.3 Algoritmi di visita
Sono algoritmi che consenton di analizzare tutti i nodi dell'albero in un ordine definito. La visita di un albero consiste nel seguire una rotta di viaggio che consenta di esaminare ogni nodo esattamente una volta.

6.3.1 Visita in preordine
Si applica ad un albero non vuoto e richiede prima l'analisi della radice e poi, la visita, effettuata con lo stesso metodo dei due sottoalberi, prima il sinistro e poi il destro.

6.3.2 Visita in postordine
Prima la visita dei sottoalberi, prima il sinistro e poi il destro, in seguito l'analisi della radice dell'albero.

6.3.3 Visita simmetrica
Richiede prima la visita del sottoalbero sinistro, poi l'analisi della radice, poi la visita del sottoalbero destro.



Preordine: 1, 2, 4, 5, 8, 9, 3, 6, 7
Postordine: 4, 8, 9, 5, 2, 6, 7, 3, 1
Simmetrica: 4, 2, 8, 5, 9, 1, 6, 3, 7

6.4 Rappresentazioni

6.4.1 Rappresentazione sequenziale
Una possibile rappreesntazione è quella sequenziale mediante vettore. La radice è in prima posizione, per il generico nodo p memorizzato in posizione i, se esiste il figlio sinistro esso è memorizzato in posizione 2*i e il figlio destro in 2*i+1. Problemi: Alcune componenti del vettore non corrispondono ad alcun nodo dell'albero (in caso di albero non completo), quindi abbiamo bisogno di un valore per esprimere il “non definito”. La soluzione è quella di utirlizzare un campo per ogni componente dell'array che varrà se vero se è presente un nodo, falso altrimenti.
Comunque:
- Alberi binari non completi vengono rappresentati con spreco di memoria
- E' imposto un limite massimo per il numero di nodi nell'albero
- Le operazioni di aggiunta ed eliminazione di nodi o di sottoalberi comportano diversi spostamenti nell'array.

6.4.2 Rappresentazione collegata

Ogni valore T del tipo albero può essere rappresentato con una lista nel modo seguente:
- Se T è vuoto, la lista che lo rappresenta è la lista vuota
- Se T non è vuoto, la lista che lo rappresenta è formata da:
    - Il primo atomo che rappresenta la radice di T
    - Il secondo è una lista che rappresenta il sottoalbero sinistro di T
- Il terzo è un altra lista che rappresenta il sottoalbero destro di T

con cursori

E' possibile utilizzare un array in modo che ad ogni nodo dell'albero corrisponda una componente dell'array in cui sono memorizzate le info: nodo, riferimento figlio sin e des. Il riferimento è il valore dell'indice in corrispondenza del quale si trova la componente corrispondente. Se il figlio non esiste ha valore 0.

con puntatori

Siccome l'albero può essere visto come una lista, è possibile utilizzare i puntatori invece dei cursori, la mancanza di un figlio viene indicato col valure null. Possiamo utilizzare anche un campo padre per efficienza.

7. ALBERI N-ARI

7.1 Specifica sintattica

Tipi: albero, boolean, nodo

Operatori:

creaAlbero:	()	→	albero
alberoVuoto:	(albero)	→	boolean
insRadice:	(nodo, albero)	→	albero
radice:	(albero)	→	nodo
padre:	(nodo, albero)	→	nodo
foglia:	(nodo, albero)	→	boolean
primoFiglio:	(nodo, albero)	→	nodo
ultimoFratello:	(nodo, albero)	→	boolean
succFratello:	(nodo, albero)	→	nodo
insPrimoSottoalbero	(nodo,albero,albero)	→	albero
insSottoalbero:	(nodo,albero,albero)	→	albero
cancSottoalbero:	(nodo, albero)	→	albero

7.2 Specifica semantica

Tipi:
- albero: insieme degli alberi ordinati T = <N,A> in cui ad ogni nodo n in N è associato il livello (n)
- boolean: insieme dei valori di verità
- nodo: nsieme qualsiasi non infinito

Operatori:
- creaAlbero() = T
  POST: T = (∅, ∅) = Λ
- alberoVuoto(T) = b
  POST: b = VERO se T = Λ, FALSO altrimenti
- insRadice(u,T) = T'
  PRE: T = Λ
  POST: T' = (N,A), N = {u}, livello(u) = 0, A = ∅
- radice(T) = u
  PRE: T ≠ Λ
  POST: u è radice di T, livello(u) = 0
- padre(u,T) = v
  PRE: T ≠ Λ, u ∈ N, livello(u) > 0
  POST: v è padre di u, <v,u> ∈ A, livello(u) = livello(v) + 1
- foglia(u,T) = b
  PRE: T ≠ Λ, u ∈ N
  POST: b = VERO se non esiste un v ∈ N tale che <u,v> ∈ A e livello(v) = livello(u)+1, altrimenti b è FALSO
- primoFiglio(u,T) = v
  PRE: T ≠ Λ, u ∈ N, foglia(u,T) = FALSO
  POST: <u,v> ∈ A, livello(v) = livello(u)+1, v è primo secondo la relazione d'ordine stabilita tra i figli di u
- ultimoFratello(u,T) = b
  PRE: T ≠ Λ, u ∈ N
  POST: b = VERO se non esistono altri fratelli di u che lo seguono nella relazione d'ordine, FALSO altrimenti
- succFratello(u,T) = v
  PRE: T ≠ Λ, u ∈ N, ultimoFratello(u,T) = FALSO
  POST: v è il fratello di u che lo segue nella relazione d'ordine
- insPrimoSottoalbero(u.T,T') = T''
  PRE: T ≠ Λ, T' ≠ Λ, u ∈ N
  POST: T'' è ottenuto da T aggiungendo l'albero T' la cui radice r' è il nuovo PrimoFiglio di u
- insSottoalbero(u,T,T') = T''
  PRE: T ≠ Λ, T' ≠ Λ, u ∈ N
  POST: T'' è ottenuto da T aggiungendo l'albero T' la cui radice r' diventa il nuovo fratello che segue u nella relazione d'ordine
- cancSottoalbero(u,T) = T'
  PRE: T ≠ Λ, u ∈ N
  POST: T' è ottenuto da T togliendovi il sottoalbero di radice u, cioè u e tutti i suoi discendenti

7.3 Visite

Sia T un albero non vuoto di radice r. Se r non è foglia e ha k > 0 figli, siano T1,…,Tk. Gli ordini di visita sono:

Previsita:(preordine) Consiste nell'esaminare r e poi nell'ordine, effettuare la previsita di T1, …,Tk.

Postvisita:(postordine) Consiste nel fare prima la postvisita di T1,…,Tk e poi esaminare la radice r.

Invisita:(ord. Simmetrico) Consiste nel fare, nell'ordine la invisita di T1,…,Ti, nell'esaminare r

e poi effettuare l'invisita di Ti+1,…,Tk per un prefissato i >= 1.

7.4 Equivalenza di alberi n-ari e binari

E' sempre possibile rappresentare un albero n-ario ordinato T con un albero binario B avente gli stessi nodi e la stessa radice: in B ogni nodo ha come figlio sinistro il primo figlio in T e come figlio destro il fratello successivo in T. Le sequenze di nodi esaminati su T e B coincidono se visitati in previsita.

7.5 Rappresentazioni

7.5.1 Rappresentazione con vettore di padri

Se numeriamo i nodi di T da 1 a n, la realizzazione sequenziale più semplice consiste nell'usare un vettore che contiene per ogni nodo i (1 =< i =< n) il cursore al padre. E' piu facile visitare i nodi lungo percorsi che vanno da foglie a radice. E' invece più complesso inserire e cancellare sottoalberi.

7.5.2 Rappresentazione attraverso liste di figli

E' una variante molto usata poiché non si può associare ad ogni elemento un numero di puntatori uguale al massimo dei figli. Comprende il vettore dei nodi in cui oltre alle eventuali etichette, si memorizza il riferimento iniziale di una lista associata ad ogni nodo. Queste liste vengono dette liste dei figli. La lista associata al generico nodo contiene tanti elementi quanti sono i successori di i. Ciascun elemento è un riferimento ad uno dei successori.

7.5.3 Rappresentazione mediante lista primofiglio/fratello

Prevede la gestione di una lista e questo può essere fatto imponendo che tutti gli alberi condividano un area comune e che ogni cella contenga esattamente due cursori: uno al primofiglio ed uno al fratello successivo, è possibile prevedere anche un cursore al genitore.

7.5.4 Rappresentazione collegata mediante lista dinamica

Da un punto formale l'albero n-ario può essere rappresentato mediante liste secondo le seguenti regole:

- Se l'albero è vuoto, la lista che lo rappresenta è vuota.
- Altrimenti l'albero è composto da una radice e da k sottoalberi T1,…,Tk e la lista è fatta da K+1 elementi: il primoil primo rappresenta la radice mentre gli altri sono gli alberi T1,…,Tk. In generale, la radice dell'albero viene memorizzata nel primo elemento della lista che contiene il riferimento ad una lista di elementi, uno per ogni sottoalbero, ciascuno di questi elementi contiene a sua volta il riferimento iniziale alla lista che rappresenta il corrispondente sottoalbero. Una possibile realizzazione prevede record e puntatori, ma il record va inteso con varianti: per esempio, si può prevedere un record con tre campi, uno per la parte informazione e due per i puntatori, per ogni record sarà sempre significativo uno dei campi puntatore, ma quando l'atomo rappresenta un nodo effettivo dell'albero sarà utilizzata l'etichetta e un puntatore, quando rappresenta un atomo "di servizio" saranno utilizzati due puntatori.

7.6 Realizzazione di MFSET

Un mfset è una partizione di un insieme finito in sottoinsiemi disgiunti detti componenti. E' possibile rappresentarlo mediante una foresta di alberi radicati in cui ciascun albero rappresenta una componente, le componenti iniziali di mfset sono i nodi e attraverso operazioni di fondi e trova si crea la struttura.
L'operazione fondi combina due alberi nello stesso albero, si realizza imponendo che una

delle due racidi diventi nuovo figlio dell'altra. L'operazione trova verifica se due elementi sono nel medesimo albero. Sirealizza accedendo ai nodi contenenti gli elementi e risalendo da tali nodi, attraverso i padri, fino ad arrivare alle radici.

8. CODE CON PRIORITA'

Sono un caso particolare di insieme, sugli elementi del quale è definita una relazione "=<" di ordinamento totale.

8.1 Specifica sintattica

Tipi: prioriCoda, tipoElem

Operatori:

creaPioriCoda:	()	→	prioriCoda
inserisci:	(tipoelem,prioriCoda)	→	prioriCoda
min:	(prioriCoda)	→	tipoElem
cancellaMin:	(prioriCoda)	→	prioriCoda

8.3 Specifica semantica

Tipi:
prioriCoda = insieme di code con priorità con elementi di tipo tipoElem

Operatori:
- creaPrioriCoda() = A
  POST: A = ∅
- inserisci(x,A) = A'
  POST: A' = A U {x} (se $x \in A$ allora A = A')
- min(A) = x
  PRE: $A \neq \emptyset$
  POST: $x \in A$, $x < y$ per ogni $y \in A$, $x \neq y$
- cancellaMin(A) = A'
  PRE: $A \neq \emptyset$
  POST: A' = A − {x} con x = min(A)

8.4 Rappresentazioni

8.4.1 Rappresentazione con strutture sequenziali

Si può rappresentare una coda con priorità di n elementi utilizzando strutture sequenziali come liste ordinate e liste non ordinate. Tuttavia poiché la coda con priorità è costituita da un insieme di atomi linearmente ordinati ma senza alcuna relazione strutturale sull'insieme delle posizioni, la sua rappresentazione è associata al modello dell'albero binario.

8.4.2 Rappresentazione con alberi binari

Gli elementi di una coda con priorità C possono essere memorizzati nei nodi di un albero binario B che deve avere le seguenti proprietà:
1. L'albero B è quasi perfettamente bilanciato:
   - se K è il livello massimo delle foglie, allora B ha esattamente (2^k)-1 nodi di livello minore di K
- tutte le foglie di livello K sono addossate a sinistra

2. L'albero B è parzialmente ordinato
    - Ogni nodo contiene un elemento di C che è maggiore di quello del padre

Per realizzare gli operatori si osservi che:
- min restituisce il contenuto della radice dell'albero B
- inserisci deve inserire una nuova foglia in modo da mantenere verificata la proprietà 1 e quindi far salire l'elemento introdotto fino a verificare la proprietà 2
- cancellaMin prevede la cancellazione della foglia di livello massimo più a destra, in modo da mantenere verificata la prop 1 ed il reinserimento del contenuto della foglia cancellata nell'albero partendo dalla radice e facendolo scendere in modo che l'albero verifichi anche la prop 2.

8.4.3 Rappresentazione con heap
Gli elementi idi B possono essere disposti in un vettore H (heap) nell'ordine in cui si incontrano visitando l'albero per livelli crescenti ed esaminando da sinistra verso destra i nodi allo stesso livello. In tal caso si ha che:
- H[1] è l'elemento contenuto nella radice di B
- H[2i] e H[2i+1] sono gli elementi corrispondenti al figlio sinistro e destro di H[i]

Se B contiene n elementi allora il figlio sinistro (destro) di H[i] non esiste nell'albero se e solo se $2i > n$ ($2i+1 > n$). Inoltre per la prop 2, se il figlio sinistro (destro) di H[i] esiste, allora $H[2i] > H[i]$ ($H[2i+1] > H[i]$)

9. GRAFI
Il grafo è una struttura composta da nordi e archi che rappresenta una relazione binaria sull'insieme dei nodi.

9.1 Specifica sintattica
Tipi: grafo, boolean, nodo, lista, tipoElem

Operatori:

- crea:	()	→	grafo
- vuoto:	(grafo)	→	boolean
- insNodo:	(nodo, grafo)	→	grafo
- insArco:	(nodo, nodo, grafo)	→	grafo
- cancNodo:	(nodo, grafo)	→	grafo
- cancArco:	(nodo, nodo, grafo)	→	grafo
- adiacenti:	(nodo, grafo)	→	lista
- esisteNodo:	(nodo, grafo)	→	boolean
- esisteArco:	(nodo, nodo, grafo)	→	boolean
- leggiNodo:	(nodo, grafo)	→	tipoElem
- scriviNodo:	(tipoElem, nodo, grafo)	→	grafo

9.2 Specifica semantica
Tipi:
- grafo: insieme G = (N, A) con N sottoinsieme finito di elementi di tipo "nodo" e $A \subseteq N \times N$
- nodo: insieme finito qualsiasi
- lista: lista di elementi di tipo nodo
- boolean: insieme dei valori di verità

Operatori:
- crea() = G
  POST: G = (N, A) con N = ∅ e A = ∅
- vuoto(G) = b
  POST: b = VERO se N = ∅ e A = ∅, b = FALSO altrimenti
- insNodo(n, G) = G'
  PRE: G = (N, A) e n ∉ N
  POST: G' = (N', A) con N' = N U {n}
- insArco(n, m, G) = G'
  PRE: G = (N,A), n ∈ N, m ∈ N, (n,m) ∉ A
  POST: G' = (N,A'), A' = A U {(n,m)}
- cancNodo(n,G) = G'
  PRE: G = (N,A), n ∈ N, ∄ m ∈ N tale che (n.m) ∈ A
  POST: G' = (N',A), N' = N \ {n}
- cancArco(n,m.G) = G'
  PRE: G = (N,A), n ∈ N, m ∈ N, (n,m) ∈ A
  POST: G' = (N,A'), A' = A \ {(n.m)}
- adiacenti(n,G) = L
  PRE: G = (N,A), n ∈ N
  POST: L è una lista che contiene una e una sola volta gli elementi di
  A(u) = {m ∈ N | (n,m) ∈ A}
- esisteNodo(n,G) = b
  POST: b = VERO se n ∈ N
- esisteArco(n,m,G) = b
  PRE: G = (N,A), n ∈ N, m ∈ N
  POST: b = VERO se ∃ (n,m) ∈ A

9.3 Rappresentazioni

9.3.1 Rappresentazione con matrice di adiacenza
La più semplice rappresentazione utilizzia una matrice N x N, E = [eij] tale che eij = 1 nel caso (i,j) ∈ A, mentre eij = 0 se (i,j) ∉ A. Se il grafo è pesato, nella matrice si utilizzano i pesi degli archi al posto degli elementi binari. E' possibile utilizzare la stessa rappresentazione per un grafo non orientato, ne risulterà una matrice simmetrica rispetto alla diagonale principale (eij = eji)

9.3.2 Rappresentazione con matrice d'incidenza
Un grafo G = (N,A) èuò anche essere rappresentato mediante una matrice (N x M), B = [bik] nella quale ciascuna riga rappresenta un nodo e ciascuna colonna un arco. Per un grafo non orientato bik = 1 se l'arco j-esimo è incidente nel nodo i, 0 altrimenti.
Nel caso di grafi orientati o diretti, il generico elemento bij di B diviene:
a) +1 se l'arco j-esimo entra nel nodo i
b) -1 se l'arco j-esimo esce dal nodo i
c) 0 altrimenti

In questo caso però, dato un nodo, non è facile ricavarne l'insieme di adiacenza. Per calcolare A(u) è necessario scandire la riga u di B alla ricerca delle colonne k tale che buk = -1 e per ogni colonna k scandire l'indice di riga i tale che bik = +1

9.3.3 Rappresentazione con vettore di adiacenza: grafo orientato

E' possibile rappresentare il grafo (N,A) con due vettori, il vettore NODI e il vettore ARCHI. Il vettore Nodi è formato da N elementi e NODI(i) contiene un cursore alla posizione di ARCHI a partire dal quale è memorizzato A(i).

grafo non orientato

E' necessario rappresentare ogni arco due volte, se i frafi sono etichettati sui nodi o sugli archi i pesi possono essere memorizzati in vettori PESINODI(n) e PESIARCHI(m)

9.3.4 Rappresentazione con liste di adiacenza

E' possibile anche utilizzare un vettore di nodi A ed n liste. Una generica componente A(i) del vettore è il puntatore alla lista i-esima in cui sono memorizzati i nodi adiacenti a i.

matrice di adiacenza estesa

Contiene campi aggiuntivi:
label: etichetta del nodo
mark: 0 se il nodo è stato rimosso, 1 se è presente
archi: la somma degli archi entranti ed uscenti