Dindo Stefano Matricola 508892 Progetto 4

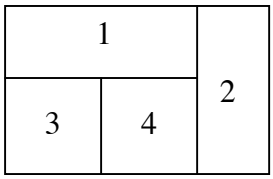
Il progetto riguarda l'uso dei trie per la memorizzazione efficiente di dataset transazionali da analizzare in applicazioni di dataming. La realizzazione del trie si articola mediante sei classi java: *Item, FrequencyVector, HashTable, Quicksort, Node, Trie, TrieBuild*.

ANALISI DELLE CLASSI:

- *Item:* la classe item costruisce un oggetto con le caratteristiche di valore e frequenza. I metodi presenti in questa classe sono semplici metodi per:
  - restituire il parametro salvato nella variabile come i metodi: tDato(), tFreqeuncy()
  - inserire il dato o la frequenza dall'esterno come i metodi: insertDato(), insertFrequency()
  - aggiornare la frequenza come il metodo adFrequency che aumenta la frequenza di una unità.

- *FrequencyVector:* ha lo scopo di implementare un vettore di Item mediante l'uso del metodo insert che riceve come parametro di ingresso un intero, e come parametro di uscita un array di Item.
  - Il metodo insert verifica che l'intero ricevuto come parametro di ingesso non sia presente nel vettore. Se è presente aggiorna la sua frequenza e termina il ciclo restituendo il vettore di item. In caso non fosse presente inserisce il nuovo elemento nell'ultima posizione indicata dall'indice, settando la frequenza a 1.
  - Anche questa classe prevede alcuni metodi necessari per accedere al contenuto del vettore.

- *Nodo:* la seguente classe implementa la struttura dei nodi del trie. I nodi sono costituiti da quattro parti:
  1) Contiene un item
  2) Contiene la referenza al fratello
  3) Contiene la referenza al figlio
  4) Contiene la referenza alla tabellahash dei figli



  All'interno di questa classe sono presenti i soliti metodi per accedere al contenuto delle varie parti del nodo quali il valore dell'item e della frequenza. Il metodo aggiuntivo riguarda la ricerca tra i fratelli, necessario nella classe Trie per verificare se un item è già presente tra i fratelli. Questo metodo ha come parametro di ingresso un item, scorre tutti i fratelli e se lo trova restituisce il nodo contenente l'item cercato, altrimenti restituisce l'ultimo della lista.

- *HashTable:* implementa i metodi principali per la realizzazione di un trie mediante hash table, con risoluzione delle collisioni mediante *SEPARATE CHAINING*. I metodi principali presenti sono:
  - searchItem: questo metodo ha come parametro di ingresso un item v, mentre ha come parametro di ritorno un nodo, con referenza a null se non è presente un nodo contenente l'item v altrimenti restituisce il nodo contenente v.
  - Inserth: questo metodo riceve come parametro di ingresso un nodo e prima di procedere con l'inserimento verifica che la Load Factors non superi il limite massimo di 1/2. Se la supera tale valore esegue il metodo di resize, altrimenti calcola la chiave del bucket in cui inserire il nuovo nodo. Se la locazione del bucket è vuota inserisce il nodo, altrimenti, se il bucket contiene già dei nodi scorre tutta la lista per inserire il nodo alla fine.

- *QuickSort:* questo algoritmo si basa sulla strategia "dividi e conquista". Per ordinare una porzione a[from]…..a[to] dell'array a, si dispongono dapprima gli elementi in modo tale che nessun elemento della porzione a[from]…a[p] sia maggiore di elementi dell'altra porzione a[p+1]…a[to]. Questo passo viene detto *suddivisione* della porzione. Al passo successivo viene

ordinata ciascuna porzione di a applicando ricorsivamente lo stesso algoritmo alle due porzioni. Ciò ordina l'intera porzione originaria. All'interno della classe QuickSort sono presenti anche i metodi orderDecrescente e orderCrescente. Il primo prevede l'ordinamento degli Item in base alla frequenza decrescente(dall'item di frequenza maggiore a quello di frequenza minore), mentre il secondo prevede l'ordinamento contrario, cioè in ordine crescente.

- *Trie:* questa classe è la più importante in quanto implementa i due metodi principali per la costruzione del trie mediante struttura linkata e mediante hash table.
  - Il metodo *insert* è il metodo che implementa la costruzione dell'albero mediante struttura linkata. Questo metodo riceve i seguenti parametri di ingresso:1)un array di item, che contiene la stringa letta dal file e ordinata per frequenza; 2)un intero i; 3) un nodo n. Ad ogni iterazione l'algoritmo prevede di verificare se il nodo da cui parte la generica iterazione ha figli. Se il nodo non ha figli costruisce un nuovo nodo contenente l'item a[i], altrimenti verifica se il figlio contiene l'item cercato. Se lo contiene, passa all'item a[i+1] oppure cerca tra i fratelli del nodo n. Se trova l'item tra i fratelli passa ad a[i] successivo o lo inserisce tra i fratelli del nodo n nella posizione corretta in quanto i fratelli devono essere ordinati in base al valore dell'item contenuto.
  - Il metodo createHash è il metodo che implementa la costruzione dell'albero mediante tabella hash. CreateHash riceve gli stessi parametri di ingresso del metodo insert. Questo metodo verifica se la referenza alla tabella hash dei figli del nodo n è vuota e in tal caso crea una tabella hash e inserisce il nodo nel bucket corretto mediante il metodo insertSonHash. Se è presente una tabella hash contenente i figli di n verifica se l'item a[i] è presente mediante il metodo searchItem. Se è presente passa ad a[i+1], altrimenti inserisce il nuovo nodo nella posizione del bucket corretta.

- *TrieBuild:* è la classe contenente il main che ha il compito di unire tutte le classi precedenti per costruire lo standard trie. La costruzione del trie prevede:
  - una scansione del file contenente i dati di ingresso per calcolare la frequenza di ogni singolo intero(il calcolo della frequenza viene svolto dai metodi contenuti nella classe FrequencyVector).
  - Scelta della modalità dei costruzione dell'albero che prevedeva quattro opzioni:
    k=0 standard trie mediante ordinamento decrescente
    k=1 standard trie mediante ordinamento crescente
    k=2 standard trie mediante ordinamentio decrescente con hash table con risoluzione delle collisioni mediante *separate chaining*
    k=3 standard trie mediante ordinamentio crescente con hash table con risoluzione delle collisioni mediante *separate chaining*

  La struttura dell'algoritmo di ogni scelta è uguale e prevede subito l'operazione di ordinamento del vettore v in base all'ordinamento previsto. Ad una seconda lettura del file di input, ogni riga letta veniva ordinata per frequenza, passata al metodo insert(contenuto nella classe trie) che provvede a costruire il ramo dell'albero ordinato per frequenza. Solamente dopo aver costruito il ramo passa alla lettura della riga successiva.

RISULTATI OTTENUTI: (Processore Athlon 1Ghz, Ram 256MB, Windows XP)

Modalità	**Tempo di esecuzione con calcolo della frequenza [*ms*]**	**Tempo di esecuzione senza calcolo della frequenza [*ms*]**
0	6510	3796
1	2577190	254927
2	6359	4116
3	244531	242288