Statistiche d'ordine dinamiche: ABR vs Lista Ordinata

Laboratorio di Algoritmi
2023/2024

Angeli Giovanni

Indice

Elenco delle figure

1 Introduzione

Lo scopo di questo esercizio è di confrontare varie implementazioni di statistiche d'ordine dinamiche attraverso strutture dati differenti ovvero lista ordinata e Alberi Binari di Ricerca (ABR) per capire quale delle alternative è più conveniente da utilizzare. Verranno eseguiti quindi dei test per valutare le prestazioni di OS-SELECT e OS-RANK, senza aumentare l'ABR con il campo size, per analizzarne vantaggi e svantaggi.

2 Spiegazione teorica del problema

In questa sezione si discuteranno le varie implementazioni mettendo a confronto la complessità temporale dei metodi citati sopra, che abbiamo studiato durante il corso di Algoritmi e Strutture Dati.

2.1 Strutture Dati

- *Lista Ordinata:* È una struttura dati in cui gli elementi sono collegati tra loro tramite puntatori in modo che siano disposti in ordine crescente o decrescente. Ogni nodo contiene due campi uno per memorizzare il valore dell'elemento e un puntatore al successivo nodo nella sequenza.
- *ABR:* È una struttura dati che ha un nodo come "radice" e ciascun nodo può avere fino a 2 figli (se non sono presenti il nodo si dice "foglia"). L'albero rispetta le seguenti regole:
  1. Il sotto-albero sinistro di un nodo x contiene soltanto i nodi con chiavi minori della chiave del nodo;
  2. Il sotto-albero destro di un nodo x contiene soltanto i nodi con chiavi maggiori della chiave del nodo x;
  3. Il sotto-albero destro e il sotto-albero sinistro devono essere entrambi due alberi binari di ricerca.

2.2 Algoritmi analizzati

In sintesi vediamo le due operazioni aggiunte alle strutture dati per implementare le statistiche d'ordine dinamiche:

- *OS-Select(x, i):* É una funzione che prende in ingresso un nodo x, un intero i e ritorna il puntatore al nodo che contiene l'i-esima chiave più piccola del sottoalbero con radice in x.
- *OS-Rank(T, x):* Questa funzione prende in ingresso la struttura dati T, un nodo x e ritorna la posizione (rango) di x nell'ordine lineare determinato da un attraversamento inorder di T.

2.3 Assunti e ipotesi

L'obiettivo è quello di implementare le due operazioni in un ABR senza aumentare la struttura dati. Per fare questo dobbiamo quindi eseguire un attraversamento in-order e contare quanti nodi sono stati trovati. Dalla teoria sappiamo che attraversare i nodi dell'albero scorrendo lungo tutta la struttura ha una complessità proporzionale al numero di nodi nell'albero, che è O(n). Questo si verifica sia al caso peggiore, che al caso medio

Nella lista ordinata in ordine crescente invece richiederanno un tempo costante se l'elemento da selezionare è in testa alla lista. Se l'elemento è in coda dovremmo scorrere l'intera lista fino al nodo desiderato e quindi la complessità sarà O(n), dove n è il numero di elementi nella lista. Nel caso medio dobbiamo scorrere circa metà della lista, quindi la complessità sarà O(n/2), che si semplifica a O(n). Di seguito nelle Tabelle 1 e 2 vengono mostrate le complessità temporali delle due operazioni nelle due diverse strutture dati

	Complessità al caso peggiore	**Complessità al caso medio**
Lista ordinata	$O(n)$	$O(n)$
ABR	$O(n)$	$O(n)$

Tabella 1: *Complessità di OS-SELECT*

	Complessità al caso peggiore	**Complessità al caso medio**
Lista ordinata	$O(n)$	$O(n)$
ABR	$O(n)$	$O(n)$

Tabella 2: *Complessità di OS-RANK*

Secondo quando detto in precedenza confronteremo le due strutture dati nei loro casi medi, con l'obbiettivo di verificare le due implementazioni di statistiche d'ordine dinamiche

3 Documentazione del codice

3.1 Schema del contenuto e delle interazioni fra i moduli

Come vediamo in Figura 1 le strutture dati necessarie per lo svolgimento degli esperimenti sono state implementate attraverso le classi LinkedList e ABR, che rappresentano una lista collegata ordinata e un albero binario di ricerca. Per l'utilizzo di queste due strutture dati sono fondamentali i nodi e quindi le classi Node e NodeABR. Node rappresenta il nodo della lista collegata e quindi come descritto nel capitolo 2.1 ha come attributi key, cioè il valore, e next che è un puntatore al nodo successivo. NodeABR è utilizzata come nodo per l'albero binario di ricerca e ha come key, left, right e parent. Questi ultimi tre sono puntatori al figlio sinistro, destro e al padre rispettivamente. I nodi si aggregano con le relative strutture dati(LinkedList e ABR), il primo per tenere traccia dell'elemento di testa della lista, il secondo della radice. Possiamo vedere che le classi Node e NodeABR hanno un vincolo di aggregazione ricorsivo di moltiplicita 1 e 3. Questo perchè un oggetto della classe Node deve avere un istanza del nodo successivo, per la classe NodeABR invece deve avere al suo interno le istanze del nodo padre e dei nodi figli destro e sinistro

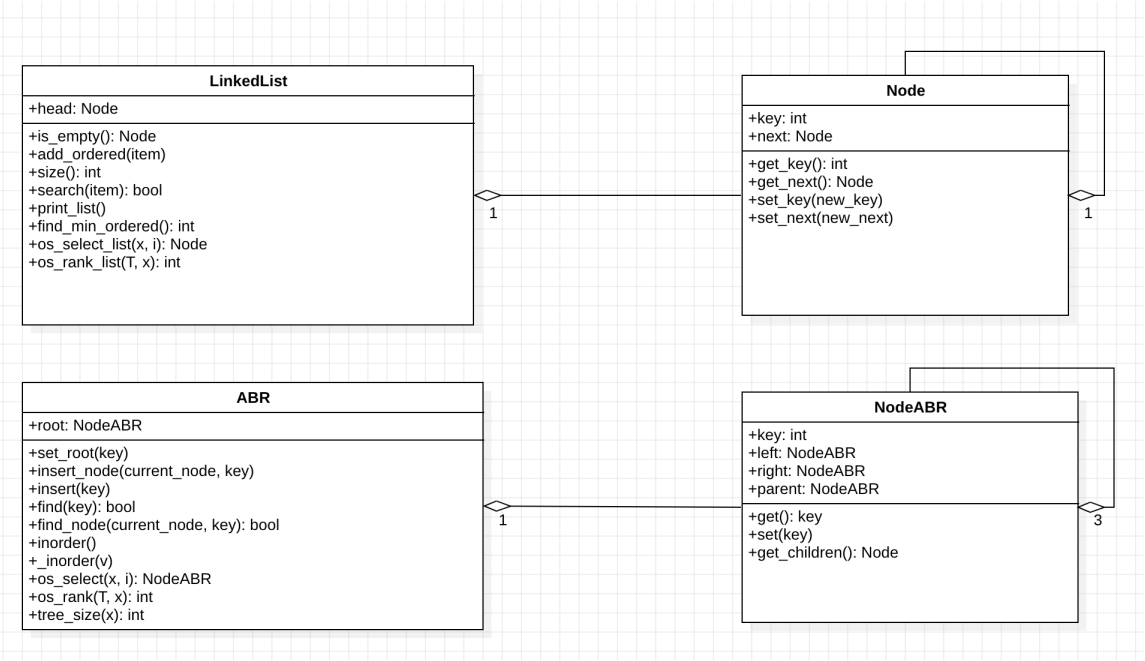

Figura 1: *Diagramma classi LinkedList e ABR*

3.2 Analisi delle scelte implementative

Il metodo *add_ordered* è stato utilizzato per mantenere la lista collegata con puntatori ordinata ad ogni inserimento.

3.3 Descrizione dei metodi implementati

In questa sezione vengono elencati e descritti brevemenți i metodi visti prima:

- **Node:**
  - get_key(): Ritorna l'attributo key
  - get_next(): Ritorna il puntatore al nodo successivo
  - set_key(new_key): Modifica l'attributo key con il valore del parametro new_key
  - set_next(new_next): Modifica l'attributo next con il puntatore del parametro new_next
- **NodeABR:**
  - get(): Ritorna l'attributo key
  - set(key): Modifica l'attributo key con il valore del parametro key
  - get_children(): Ritorna un array di nodi figli(sinistro e destro)
- **LinkedList:**
  - is_empty(): Controlla se la lista è vuota. Ritorna True se head è None altrimenti False
  - add_ordered(item): Attraversa la lista finché non trova il punto di inserimento dell'elemento e lo inserisce mantenendo l'ordine
  - size(): ritorna la dimensione della lista attraversandola e contandone gli elementi
  - search(item): Scorre la lista finché non trova l'elemento item. Ritorna True se lo trova, altrimenti False
  - print_list(): Stampa tutti gli elementi della lista
  - find_min_ordered(): Ritorna il valore minimo nella lista ordinata. Se la lista è vuota, ritorna None
  - os_select_list(x, i): Restituisce il nodo alla posizione i partendo dal nodo x
  - os_rank_list(T,x): Restituisce il rango del nodo x nella lista. Se il nodo non è presente nella lista, restituisce None

- **ABR:**
  - set_root(key): Instanzia un oggetto NodeABR, con valore key, su root
  - insert_node(current_node, key): Attraversa ricorsivamente l'albero fino a che non trova una foglia libera. Quindi crea un istanza di NodeABR con valore key
  - insert(key): Se root è Nil chiama il metodo set_root(key), altrimenti chiama il metodo insert_node(current_node, key)
  - find(key): Ritorna il valore di find_node(self.root, key)
  - find_node(current_node, key): attraversa l'albero ricorsivamente, se trova un nodo con valore key allora ritorna TRUE altrimenti FALSE
  - inorder(): Attraversa ricorsivamente l'albero "in-order", stampando i valori dei nodi
  - _inorder(v): Funzione di supporto per inorder(), attraversa ricorsivamente i sottoalberi sinistro e destro di un nodo, stampando il valore del nodo corrente
  - os_select(x, i): Ritorna il nodo di rango i nell'albero con radice in x, attraversando l'albero in modo ricorsivo
  - os_rank(T, x): Calcola e ritorna il rango del nodo x nell'albero T, risalendo il percorso verso la radice e considerando i ranghi dei nodi nei loro sottoalberi.
  - tree_size(x): Calcola e ritorna la dimensione dell'albero con radice nel nodo x, eseguendo un attraversamento in-order

4 Esperimenti effettuati ed Analisi dei risultati

4.1 Specifiche della piattaforma di test

Le specifiche hardware della macchina sono:

- **CPU:** Apple M1 8-core
- **RAM:** Micron 8 GB LPDDR4
- **SSD:** Apple 256GB

La piattaforma in cui il codice è stato scritto e testato è l'IDE PyCharm 2023.3.3 (Community Edition). Il sistema operativo è MacOS sonoma 14.2

4.2 Esperimenti condotti

I dati utilizzati per gli esperimenti sono liste ordinate e ABR con dimensioni crescenti, riempiti con valori interi casuali in un range da 0 alla dimensione delle strutture dati. Viene poi calcolato il tempo di esecuzione ogni 250 elementi inseriti.

Ogni esperimento viene ripetuto 5 volte, dopo di che viene calcolata la media dei risultati. I dati vengono poi mostrati in tabelle e grafici generati dai test.

4.3 Misurazioni

Per calcolare i tempi di esecuzione e la media degli esperimenti è stata utilizzata la libreria **timeit** di python. (Esempio in Figura 2) Dove gli argomenti di timeit() sono:

- lambda: struttura_dati.algoritmo(): è una funzione lambda usata per creare un contesto di esecuzione per timeit, la quale chiama l'algoritmo implementato sulla struttura dati
- number=5 che specifica che la funzione verrà eseguita 5 volte e verrà calcolato il tempo medio di esecuzione

Questa restituisce il tempo medio di un esperimento eseguito sull'algoritmo eseguito per 5 volte.

```
# calcolo tempi: lista ordinata selezionando elemento pù grande
select_list_time = timeit.timeit(lambda: linked_list.os_select_list(linked_list.head, size), number=5)
rank_list_time = timeit.timeit(
    lambda: linked_list.os_rank_list(linked_list.os_select_list(linked_list.head, size)), number=5)
select_list_times.append(select_list_time)
rank_list_times.append(rank_list_time)
```

Figura 2: *Esempio uso timeit*

4.4 Risultati degli esperimenti

In questa sezione vengono mostrati i risultati, sia in forma tabellare che con grafici. Le tabelle mostrano la relazione fra il numero di elementi delle strutture dati ed il tempo impiegato ad eseguire i due algoritmi. Nei grafici vediamo invece l'andamento nel tempo.

4.4.1 OS-Select

Dai test sull' OS-Select sono stati estratti i dati presenti nella Tabella 3 (per le liste) e nella Tabella 4 (per gli ABR). In entrambi i casi i tempi medi di esecuzione sembrano crescere linearmente rispetto al numero degli elementi nelle due strutture dati. Questo è confermato dai Grafici 3 e 4. Attraverso i tempi di esecuzione si nota una minima differenza tra i tempi di esecuzione nel caso delle liste rispetto a quella degli ABR ma è poco rilevante ai fini dei test

OS-Select Lista ordinata

Elementi	Tempo (s)
250	$1.038 * 10^{-4}$
500	$2.247 * 10^{-4}$
750	$3.382 * 10^{-4}$
1000	$4.328 * 10^{-4}$
1250	$5.593 * 10^{-4}$
1500	$6.580 * 10^{-4}$
1750	$7.649 * 10^{-4}$
2000	$8.798 * 10^{-4}$

Tabella 3: *Tempo di esecuzione OS-Select su lista ordinata all'aumentare del numero di elementi*

OS-Select ABR

Elementi	Tempo (s)
250	$2.292 * 10^{-4}$
500	$4.800 * 10^{-4}$
750	$6.369 * 10^{-4}$
1000	$8.593 * 10^{-4}$
1250	$1.059 * 10^{-3}$
1500	$1.295 * 10^{-3}$
1750	$1.552 * 10^{-3}$
2000	$1.761 * 10^{-3}$

Tabella 4: *Tempo di esecuzione OS-Select su ABR all'aumentare del numero di elementi*

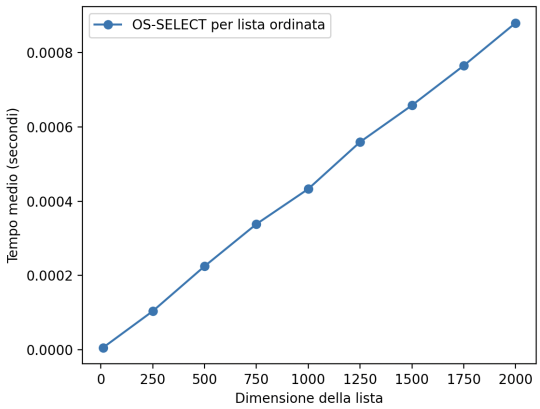

Figura 3: *Grafico Tabella 3: tempo impiegato al variare del numero di nodi*

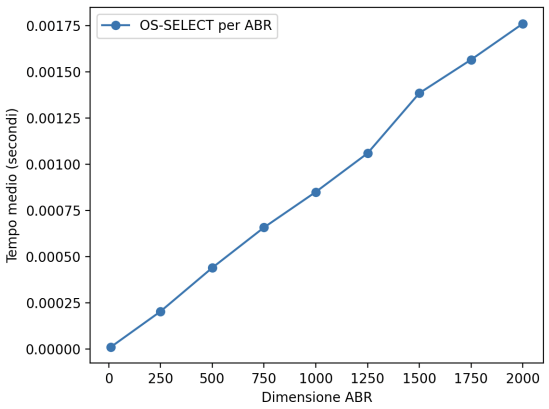

Figura 4: *Grafico Tabella 4: tempo impiegato al variare del numero di nodi*

4.4.2 OS-Rank

Anche per OS-Rank i risultati sono simili ed evidenziano un andamento lineare del tempo come possiamo vedere nei Grafici 5 e 6. Si può notare un leggero aumento nei tempi delle Tabelle 5 e 6 rispetto a quelle precedenti ma potrebbe essere dovuto ad un rallentamento della macchina durante l'esecuzione dei test o imprecisioni durante le misurazioni dei tempi

OS-Rank Lista ordinata

Elementi	Tempo (s)
250	$2.009 * 10^{-4}$
500	$4.402 * 10^{-4}$
750	$6.969 * 10^{-4}$
1000	$8.946 * 10^{-4}$
1250	$1.106 * 10^{-3}$
1500	$1.366 * 10^{-3}$
1750	$1.579 * 10^{-3}$
2000	$1.802 * 10^{-3}$

Tabella 5: *Tempo di esecuzione OS-Rank su lista ordinata all'aumentare del numero di elementi*

OS-Rank ABR

Elementi	Tempo (s)
250	$4.028 * 10^{-4}$
500	$8.573 * 10^{-4}$
750	$1.282 * 10^{-3}$
1000	$1.673 * 10^{-3}$
1250	$2.270 * 10^{-3}$
1500	$2.528 * 10^{-3}$
1750	$2.958 * 10^{-3}$
2000	$3.485 * 10^{-3}$

Tabella 6: *Tempo di esecuzione OS-Rank su ABR all'aumentare del numero di elementi*

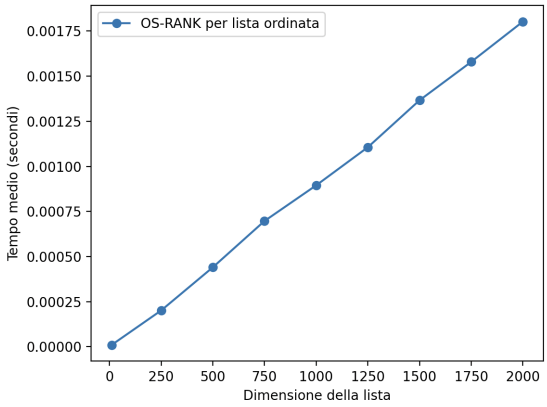

Figura 5: *Grafico Tabella 5: tempo impiegato al variare del numero di nodi*

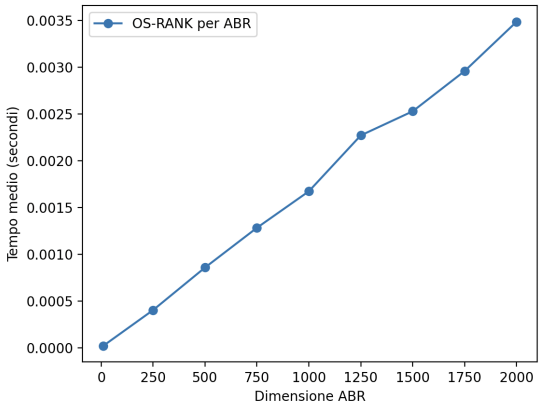

Figura 6: *Grafico Tabella 6: tempo impiegato al variare del numero di nodi*

5 Conclusioni

I risultati dei test confermano le ipotesi iniziali e quindi una complessità temporale O(n) e quindi possiamo concludere che non ci sono differenze tra utilizzare ABR o liste ordinate per implementare statistiche d'ordine dinamiche, a meno che non aumentiamo la struttura dati ABR aggiungendo un campo size che renderebbe costante (O(1)) il calcolo delle dimensioni dell'albero