Damian Ariel Dominella - 277732
Oscar Gabellini - 280525

Algoritmi e strutture dati

Progetto sessione autunnale 2016/2017

Docente: Prof. Valerio Freschi

Indice

Specifica del problema

Si supponga di dover progettare un programma per la gestione di un supermercato di ferramenta. Scrivere un programma ANSI C che esegue le seguenti elaborazioni:

1. Acquisisce un file di testo, il cui formato prevede un certo numero di righe, ognuna delle quali contiene: nome del prodotto, codice identificativo del prodotto (4 caratteri), costo unitario, numero di pezzi in magazzino (separati da tabulazione oppure da spazio). Ad esempio:

   ```
   Chiodo_L_2.5 N924 1.20 300
   Martello H610 12.50 127
   Cacciavite S234 4.55 432
   ```

2. Inserisce i dati in una opportuna struttura dati ad albero.

3. Permette all'utente di scegliere ed effettuare le seguenti operazioni:

   - Inserimento di nuovi input da tastiera
   - Cancellazione di dati esistenti, selezionandoli opportunamente da tastiera
   - Stampa a monitor dell'elenco ordinato dei dati sulla base di una delle 4 chiavi, a scelta dell'utente.

Per quanto riguarda l'analisi teorica si deve fornire la complessità corrispondente ad ognuna delle seguenti operazioni: inserimento, cancellazione, stampa ordinata.

Oltre all'analisi teorica della complessità degli algoritmi implementati nel programma si deve effettuare uno studio sperimentale degli stessi. In particolare, si deve operare generando casualmente un numero N di righe da fornire in input al programma. L'analisi sperimentale deve quindi valutare la complessità al variare del parametro N per le fasi di: inserimento, cancellazione, stampa ordinata.

Analisi del problema

Analisi degli input:

L’input del programma consiste in un file di testo contenente una lista di articoli formattata, come da specifiche, nel seguente modo:

nome id prezzo quantità
nome id prezzo quantità
…

Si possono considerare come altri tipi di input le scelte effettuate dall’utente (stampa, inserimento, cancellazione) dato che vanno ad alterare l’output del programma.

Analisi degli output:

L’output del programma varia in base alla scelta dell’utente. In ogni caso l’unico vero e proprio output riguarda la stampa ordinata dei dati, con ordinamento basato su una delle 4 chiavi (nome, id, prezzo, quantità) a scelta dell’utente nel caso in cui l'azione primaria sia appunto la stampa degli elementi, basata sull’identificativo invece nel caso in cui la stampa avvenga dopo azioni quali inserimento ed eliminazione dei dati, come accertamento della buona riuscita dell’operazione.

Relazioni intercorrenti fra input ed output:

La relazione fra input ed output è basata sulla scelta dell’operazione da eseguire, ecco i casi possibili:

Input	Processo	Output
Visualizzazione dati + chiave ordinamento	Lettura dell’albero ordinata	Stampa ordinata dell’albero
Creazione di un nuovo dato	Inserimento nuovo nodo nell’albero	Stampa ordinata dell’albero con il nuovo elemento aggiunto
Cancellazione di un dato	Ricerca del nodo da rimuovere ed eliminazione dall’albero	Stampa ordinata dell’albero priva dell’elemento rimosso

Progettazione dell’algoritmo

Strutture dati

Analizzando le specifiche del problema, si nota la richiesta dell’utilizzo di una struttura dati ad albero, la quale è nota per essere una struttura gerarchica. Al fine di poter implementare una stampa ordinata per quattro chiavi differenti utilizzando questo tipo di struttura, è stato necessario l’utilizzo di **quattro alberi binari**, ognuno dei quali appartenente allo stesso tipo di struttura, ma ordinato su chiavi diverse (nome, id, prezzo, quantità).

Oltre alla struttura dati ad albero necessaria per la disposizione dei dati, è stata definita un’altra struttura per definire gli **articoli**. Si tratta di una semplice struttura dati contente i seguenti campi:

- Codice identificativo (id)
- Nome dell’articolo
- Prezzo
- Quantità in magazzino

A questo punto, i **nodi** che andranno a formare l’albero binario saranno formati da: puntatore al figlio destro, puntatore al figlio sinistro e informazione, che non sarà altro che un oggetto di tipo **articolo**.

Caricamento dati da file

La prima operazione da eseguire è il caricamento dei dati (lista di articoli presa da un file di testo esterno) all’interno della struttura dati ad albero. Vista la formattazione dei dati all’interno del file, che divide gli articoli riga per riga, viene creato un oggetto di tipo **articolo** per ognuna di esse. Questo oggetto poi verrà inserito opportunamente nell’albero. Siccome, come precedentemente spiegato, vengono adoperati quattro alberi, l’operazione di inserimento verrà ripetuta quattro volte.

Interazione con l'utente

Dato che le specifiche implicano la possibilità da parte dell'utente di effettuare diverse operazioni riguardo alla manipolazione dei dati, è stata ritenuta opportuna l'implementazione di un **menu** per permettere all'utente di effettuare le scelte indicate nelle specifiche (visualizzazione dati, inserimento, cancellazione).
Per evitare errori, l'acquisizione della scelta viene validata attraverso opportuni controlli.

- **Visualizzazione dati:** prima di stampare i dati, viene richiesta una chiave di ordinamento[1] (id, nome, quantità, prezzo). Una volta inserita e validata la chiave, viene elaborato l'albero ad essa corrispondente.

- **Inserimento dati:** per quanto riguarda l'inserimento di nuovi record, si assume che i campi **ID** e **nome** siano univoci, per tanto all'inserimento di questi ultimi viene effettuata una ricerca per verificare che non siano presenti record esistenti con tali valori. Questo tipo di controllo non viene effettuato per i campi **quantità** e **prezzo**. Inoltre, per quanto riguarda l'inserimento dell'ID, viene anche verificato che la sua lunghezza sia di 4 caratteri (come da specifiche). Una volta effettuati tutti gli accertamenti, si procede in modo analogo all'inserimento durante il caricamento dei dati. Ovvero si crea prima un oggetto di tipo **articolo**, che viene successivamente inserito nei 4 alberi.

- **Eliminazione dati:** per quanto concerne invece la rimozione dei dati, viene richiesto all'utente di inserire l'ID dell'elemento da rimuovere. Una volta verificata la presenza di un record con tale id, si procede ad eliminare tale record da tutti e 4 gli alberi.

[1] Si precisa che l'ordinamento è sempre di tipo **ascendente**

Implementazione dell’algoritmo

Visualizzazione dati

Per quanto riguarda la visualizzazione dei dati, ovvero la stampa ordinata dell’albero binario, viene utilizzato l’algoritmo **ricorsivo** standard per la visita in ordine simmetrico dell’albero. Perciò, partendo dal nodo radice, si visita con lo stesso algoritmo il sotto-albero sinistro, poi si elabora il nodo corrente ed infine si visita con lo stesso algoritmo il sotto-albero destro. Il codice della funzione può essere schematizzato nel seguente modo:

```
if (node != null) {
     visit(node->left);
     process(node);
     visit(node->right);
}
```

Inserimento dati

La logica dell’algoritmo **ricorsivo** che viene utilizzato per l’inserimento dei dati, è la medesima per ogni tipologia di albero in cui inserire il nuovo elemento, con qualche piccola variazione per quanto riguarda l’inserimento ordinato per quantità e prezzo (dal momento che questi due campi permettono dati duplicati).

Dal momento che l’inserimento dei dati dovrà operare su 4 alberi binari, è stato scelto di parametrizzare la funzione di inserimento in base al tipo di dato per il quale ordinare (per facilità di spiegazione questo parametro da ora in poi verrà chiamato **tipo**).

Oltre a questo parametro, la funzione acquisisce il nodo radice e un oggetto di tipo **articolo** da inserire nella struttura dati.
Come prima cosa, la funzione verifica che il nodo corrente sia nullo, in tal caso, procede ad inserire l’articolo in tal nodo allocando la memoria necessaria per quest’ultimo ed inizializzando i puntatori al figlio destro e sinistro a *NULL*.

Altrimenti, nel caso in cui il nodo corrente non sia nullo, si procede confrontando la chiave corretta (basata sul parametro **tipo**).

Nel caso di **ID** e **nome**, basterà confrontare la chiave dell'oggetto articolo passato alla funzione *(A)* con la chiave dell'oggetto articolo all'interno del nodo corrente *(B)*, comprendendo solamente i casi minore di, o maggiore di, tralasciando il caso in cui questi due valori siano uguali (non è una cosa possibile dato che per scelte progettuali questi due campi sono stati definiti come univoci).
Nel caso in cui *(A)* sia minore di *(B)*, si utilizza la ricorsione passando come nodo il figlio sinistro del nodo corrente. Altrimenti, si procede in modo analogo ma con il figlio destro. Il codice della funzione può essere schematizzato nel seguente modo:

```
function insert(node, article) {
    if (node == null) {
        create_node(article);
    } else {
        if (article->key < node->article->key) {
            insert(node->left, article);
        } else if (article->key > node->article->key) {
            insert(node->right, article);
        }
    }
}
```

Nel caso in cui il parametro **tipo** sia invece **prezzo** o **quantità** l'unica modifica dell'algoritmo consiste nel fatto che invece di considerare solo i casi **minore di** o **maggiore di**, verranno considerati i casi **minore o uguale di**, e **maggiore di**. La funzione diventa quindi la seguente:

```
function insert(node, article) {
    if (node == null) {
        create_node(article);
    } else {
        if (article->key <= node->article->key) {
            insert(node->left, article);
        } else {
            insert(node->right, article);
        }
    }
}
```

Eliminazione dati

Come spiegato durante la progettazione dell'algoritmo, la funzione di eliminazione per poter operare necessita del riferimento all'articolo da rimuovere, il quale viene selezionato dall'utente attraverso l'**ID**. Ciò implica l'esistenza di una funzione per cercare i dati all'interno dell'albero binario. L'algoritmo di ricerca acquisisce come parametri il nodo radice e l'id dell'articolo cercato, poi compara l'id dell'articolo all'interno del nodo corrente con l'id cercato. Nel caso in cui siano uguali allora il nodo è stato trovato e viene ritornato. Nel caso in cui l'id cercato sia **maggiore di** quello del nodo corrente allora si utilizza la ricorsione utilizzando lo stesso algoritmo partendo dal figlio destro del nodo corrente. Altrimenti si procede in modo analogo ma partendo dal figlio sinistro. La funziona di ricerca può essere cosi schematizzata:

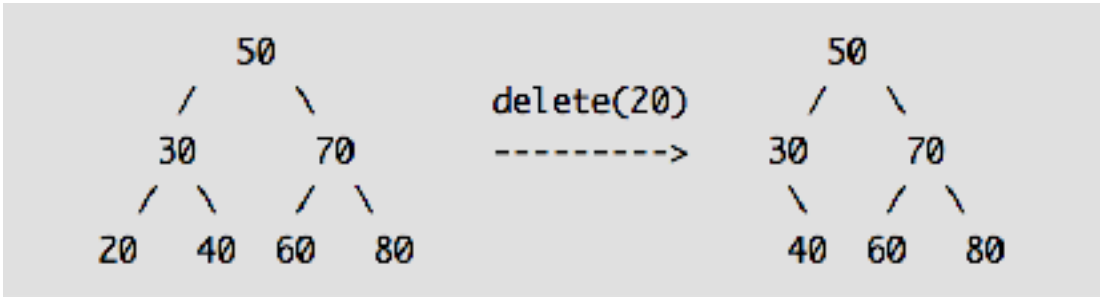
```
function search(node, id) {
    if (id == node->article->id) {
        result = node;
    } else if (id > node->article->id) {
        search(node->right, id);
    } else {
        search(node->left, id);
    }
}
```

L'algoritmo della rimozione, acquisisce come parametri il nodo radice, l'articolo e, come per la funzione di inserimento, il parametro **tipo** che sarà necessario per eliminare comparando la chiave giusta.
Inizialmente, l'algoritmo compara la chiave dell'articolo passato con la chiave dell'articolo del nodo corrente. Se quest'ultima è **minore**, allora viene utilizzata la ricorsione richiamando lo stesso algoritmo per il figlio sinistro del nodo corrente. Altrimenti, se la chiave dell'articolo del nodo corrente è **maggiore** della chiave dell'articolo passato, l'algoritmo viene rieseguito sul figlio destro del nodo corrente.
Nel caso in cui le due chiavi siano uguali, vorrà dire che il nodo da rimuovere è stato trovato. A questo punto si hanno 3 possibili casistiche:

A. Il nodo corrente non ha figli
B. Il nodo corrente ha un solo figlio
C. Il nodo corrente ha due figli

Attraverso i seguenti schemi è possibile capire come queste casistiche dovranno funzionare:

Caso A: il più semplice, il nodo non avendo figli viene semplicemente rimosso.



Caso B:

Caso C:

Per verificare il primo caso, è sufficiente controllare prima che il figlio sinistro sia nullo, ed in tal caso eliminare il nodo corrente. Altrimenti è necessario verificare che il figlio destro sia nullo, ed in tal caso eliminare il nodo corrente.
Nel secondo caso basterà copiare l'elemento del figlio non nullo nel nodo corrente ed in seguito eliminare il figlio contenente l'elemento appena copiato.
Se nessuna di queste due casistiche si verifica, si è di fronte ad un nodo con 2 figli. In questo caso si procede nel seguente modo:

1) Si ottiene il figlio con il valore minore dal sotto-albero destro[2].
2) Si copia il valore all'interno di quel nodo nel nodo corrente.
3) Si utilizza la **ricorsione** richiamando l'intero algoritmo di rimozione sul figlio destro del nodo corrente, passando come articolo da eliminare quello all'interno del nodo ricavato dal punto 1. Questo passaggio serve per eliminare il dato copiato nel nodo corrente.

La funzione di eliminazione può essere così schematizzata:

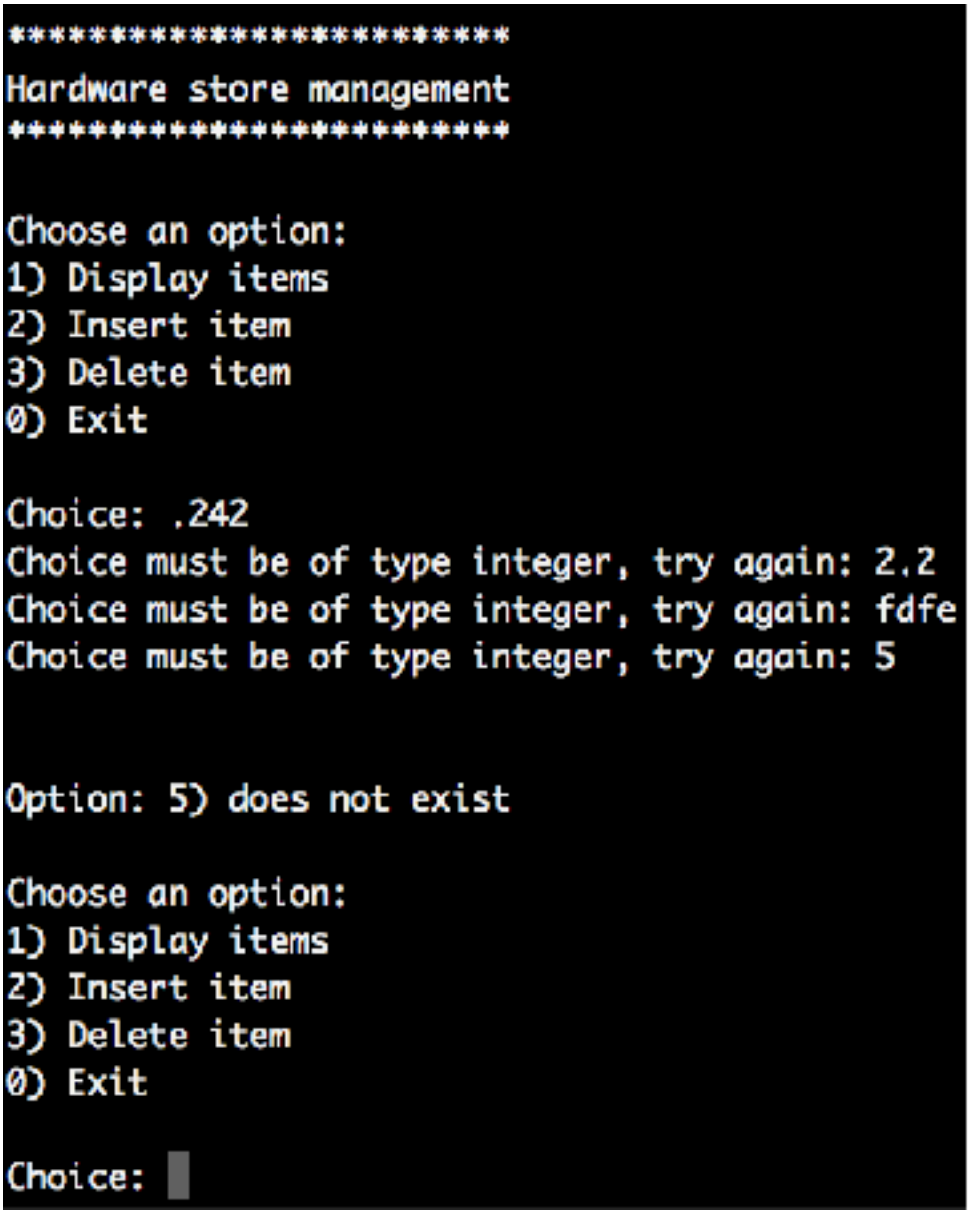
```
function delete(node, article) {
    if (article->key < node->article->key) {
        delete(node->left, article);
    } else if (article->key > node->article->key) {
        delete(node->right, article);
    } else {
        if (node->left == NULL) {
          temp = node->right;
          free(node);
          return temp;
        } else if (node->right == NULL) {
          temp = node->left;
          free(node);
          return temp;
        }
        temp = min_value_node(node->right);
        node->article = temp->article;
        delete(node->right, temp->article);
    }
}
```

[2] Si tratta di un semplice ciclo che si ferma solo quando il figlio sinistro del nodo corrente è nullo.

Testing del programma

Di seguito vengono riportati alcuni significativi test che dimostrano il funzionamento corretto del programma in base alle diverse tipologie di input proposti dall'utente.

Validazione scelta menù:



Validazione input inserimento dati:

```
Choice: 2

Insert item
------------------------------
ID: H610
A record with ID: H610 already exists, try again: AA
ID length must be of 4 characters: 12345
ID length must be of 4 characters: B890
Name: Test
Quantity: 12.4
Quantity must be of type integer, try again: 100
Price: 9.s
Price must be of type float, try again: 9.50

Record inserted successfully
------------------------------------------------------------
ID              Name                    Quantity        Price
------------------------------------------------------------
B890            Test                    100             9.50
```

Stampa ordinata in base a chiave scelta dall'utente:

ID:

```
Choice: 1

Display items, please choose a sort key
0) ID
1) Name
2) Quantity
3) Price

Sort key: 0

Data sorted by field: ID
------------------------------------------------------------
ID              Name                Quantity        Price
------------------------------------------------------------
B890            Test                100             9.50
G321            Glue                1200            0.59
H610            Hammer              127             12.50
N924            Nail_L_2.5          300             1.20
```

Nome:

```
Choice: 1

Display items, please choose a sort key
0) ID
1) Name
2) Quantity
3) Price

Sort key: 1

Data sorted by field: Name
------------------------------------------------------------
ID              Name                Quantity        Price
------------------------------------------------------------
G321            Glue                1200            0.59
H610            Hammer              127             12.50
N924            Nail_L_2.5          300             1.20
P800            Pliers              500             9.99
```

Cancellazione:

```
Choice: 3

Delete item
------------------------------------------------------------
ID              Name                Quantity        Price
------------------------------------------------------------
A101            Test                1200            4.50
G321            Glue                1200            0.59
H610            Hammer              127             12.50
N924            Nail_L_2.5          300             1.20
P800            Pliers              500             9.99
S234            Screwdriver         432             4.55
S765            Scissors            900             1.60
W089            Wrench              140             4.50
------------------------------------------------------------

ID to delete: A101

Record deleted successfully
```

Come si può notare è stato eliminato un record dove vi erano valori duplicati per quantità (1200) e prezzo (4.50). Dal seguente screenshot verifichiamo che il valore eliminato è stato quello corretto (**A101**) anche dall'albero del prezzo, pertanto è possibile affermare che la funzione di eliminazione funziona correttamente anche per valori duplicati.

```
Choice: 1

Display items, please choose a sort key
0) ID
1) Name
2) Quantity
3) Price

Sort key: 3

Data sorted by field: Price
------------------------------------------------------------
ID              Name                Quantity        Price
------------------------------------------------------------
G321            Glue                1200            0.59
N924            Nail_L_2.5          300             1.20
S765            Scissors            900             1.60
W089            Wrench              140             4.50
S234            Screwdriver         432             4.55
P800            Pliers              500             9.99
H610            Hammer              127             12.50
------------------------------------------------------------
```

Valutazione della complessità del programma - Analisi teorica

Il vantaggio di avere uno **schema** di tipo divide et impera per gli alberi è che consente di scrivere le relazioni di ricorrenza per il costo *T(n)* per ogni operazione.

Inserimento:

Ipotizzando, per comodità, che il costo della creazione di un nuovo nodo sia un valore costante **c**, e che **r** sia il numero di figli (destri o sinistri) del nodo corrente, la relazione di ricorrenza può essere così definita:

$$T(n) = \left\{ \begin{array}{cc} 1+c & n <= 1 \\ 1+T(r) & altrimenti \end{array} \right\}$$

Cancellazione:

Ipotizzando, per comodità, che il costo della rimozione di un nodo sia un valore costante **c**, che il costo del calcolo per ottenere il nodo più piccolo partendo dal figlio destro del nodo corrente sia **k** e che **r** sia il numero di figli (destri o sinistri) del nodo corrente, la relazione di ricorrenza può essere così definita:

$$T(n) = \left\{ \begin{array}{cc} 1+c & n <= 1 \\ 1+k+1+T(r) & 1 < n <= 2 \\ 1+T(r) & altrimenti \end{array} \right\}$$

Stampa ordinata:

Sapendo che il costo della stampa del singolo nodo è 1, per comodità si ipotizza che il nodo corrente abbia **r-1** nodi che discendono dal suo figlio sinistro e, quindi, **n-r** che discendono da quello destro.
Allora si può dire che:

$$T(n) = \left\{ \begin{array}{cc} 1 & n <= 1 \\ T(r-1)+1+T(n-r) & altrimenti \end{array} \right\}$$

La soluzione di tutte queste relazioni di ricorrenza è: ***O(n)***

Valutazione della complessità del programma - Studio sperimentale della complessità

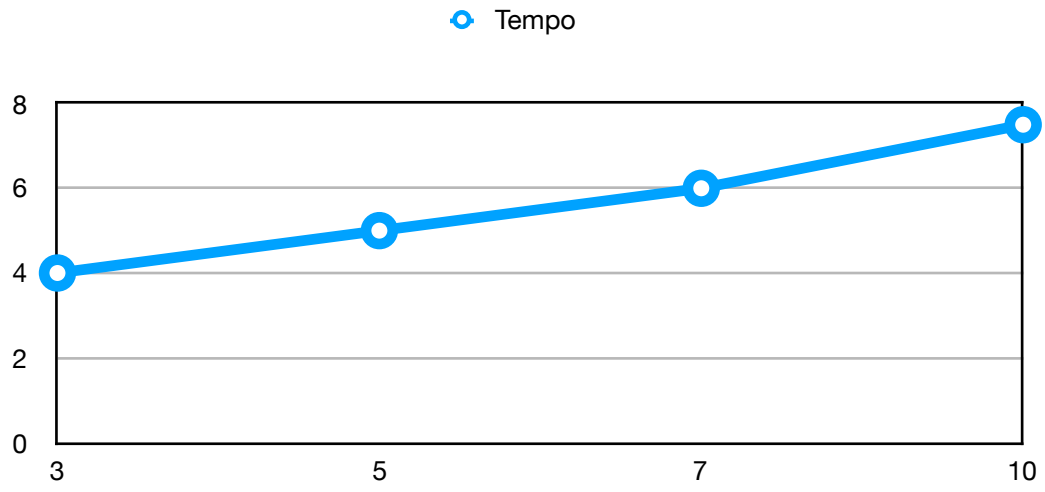
Lo studio sperimentale è stato effettuato nel seguente modo:

1. Inizio timer
2. Esecuzione
3. Fine timer
4. Ricavo del tempo di esecuzione

Al fine di ottenere risultati più corretti possibili, questi passaggi vengono effettuati 50 volte per poi ricavarne il tempo medio. [3]

Inserimento:

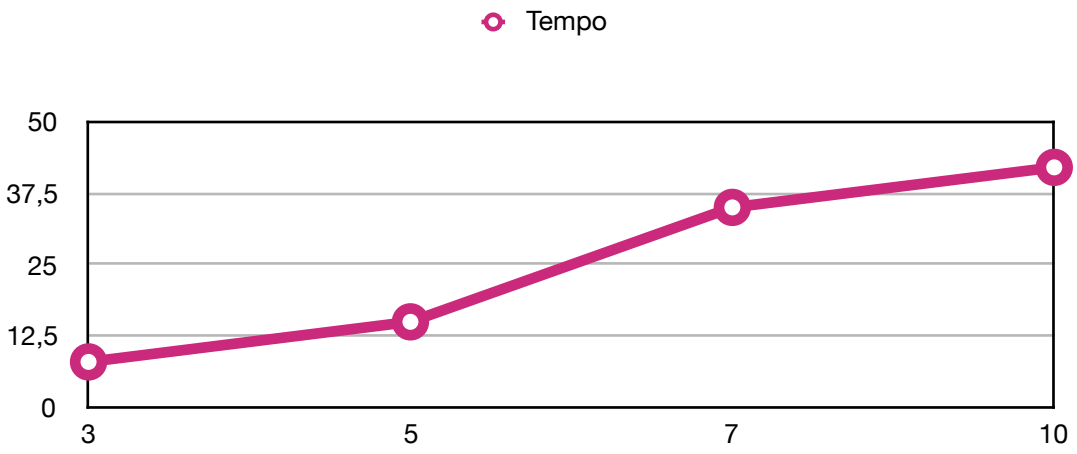
Numero articoli	Tempo medio esecuzione (ms)
3	0,004
5	0,005
7	0,006
10	0,007



[3] Si precisa che, dal momento che le funzioni di **inserimento** e **cancellazione** vengono ripetute 4 volte (una per ogni albero), il tempo di esecuzione è calcolato dopo tutte e 4 le operazioni.

Cancellazione:

Numero articoli	Tempo medio esecuzione (ms)
3	0,008
5	0,015
7	0,035
10	0,043



Stampa ordinata:

Numero articoli	Tempo medio esecuzione (ms)
3	0,042
5	0,056
7	0,071
10	0,091