

Corso di Laurea in Ingegneria e Scienze Informatiche

# B-Tree File System per piattaforme IoT

Tesi di laurea in:  
SUPERVISOR'S COURSE NAME

*Relatore*

**Marco Antonio Boschetti**

*Candidato*

**Leonardo Marcaccio**

---

---

# Abstract

Questa Tesi esplora il processo di studio, progettazione e implementazione di un prototipo di File System (FS) basato sulla struttura dati B-Tree, sviluppato per la piattaforma IoT IOtto di Onit S.p.A.

L'obiettivo principale è migliorare l'efficienza nella gestione e nel recupero dei dati, affrontando le problematiche tipiche di un FS come la scalabilità e l'ottimizzazione delle risorse.

Il lavoro comprende un'analisi dello stato dell'arte sui moderni FS, con particolare attenzione alla loro interazione con i sistemi IoT. Viene approfondito il principio di funzionamento del B-Tree, dimostrando come questa struttura dati possa essere sfruttata per realizzare una struttura performante e affidabile.

Inoltre, vengono descritte le fasi di progettazione e implementazione, evidenziando le soluzioni adottate per adattare il prototipo alle esigenze specifiche della piattaforma IOtto.

I risultati preliminari mostrano che il prototipo proposto garantisce significativi miglioramenti in termini di velocità di accesso ai dati e utilizzo delle risorse rispetto alle alternative tradizionali.

---

---

*Optional. Max a few lines.*

---

---

# Contents

<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.1.1 Onit . . . . .	1
1.1.2 IOtto . . . . .	1
1.2 Descrizione del problema . . . . .	1
1.3 Obiettivi di Tesi . . . . .	1
1.3.1 Domande . . . . .	2
1.3.2 Scopo . . . . .	2
1.4 Struttura della Tesi . . . . .	3
<b>2 State of the art</b>	<b>5</b>
2.1 L'Internet of Things . . . . .	5
2.2 File System . . . . .	6
2.3 Categorizzazione dei File System . . . . .	7
2.3.1 Single-Level Directory Systems . . . . .	7
2.3.2 Hierarchical Directory Systems . . . . .	8
2.4 Data Structures . . . . .	9
2.5 Strutture Dati Basate sui Tree . . . . .	11
2.5.1 Binary Trees . . . . .	11
2.5.2 Binary Search Trees . . . . .	12
2.5.3 Balanced Trees . . . . .	12
2.5.4 N-ary Trees . . . . .	13
2.6 B-Tree . . . . .	13
<b>3 Design and Implementation</b>	<b>15</b>
3.1 Progettazione . . . . .	15
3.1.1 Requisiti . . . . .	16
3.1.2 Architettura . . . . .	16
3.2 Fancy formulas here . . . . .	17

## CONTENTS

---

	<b>19</b>
<b>Bibliography</b>	<b>19</b>



---

# List of Figures

2.1	Esempio di Single Level Directory System . . . . .	8
2.2	Esempio di Hierarchical Directory System . . . . .	9
2.3	Esempio di Linked List . . . . .	10
2.4	Esempio di Array . . . . .	10
2.5	Esempio di Tree . . . . .	10
2.6	Esempio di Hash Table . . . . .	11

## LIST OF FIGURES

---

---

# List of Listings

listings/HelloWorld.java . . . . .	17
------------------------------------	----

## LIST OF LISTINGS

---

---

# Chapter 1

## Introduction

### 1.1 Background

#### 1.1.1 Onit

#### 1.1.2 IOtto

AGGIUNGI PRESENTAZIONE DELL'AZIENDA E DEL PRODOTTO

### 1.2 Descrizione del problema

L'attuale FS utilizzato dall'applicativo risulta non essere più ottimale nel contesto attuale, data la crescita e la mole dei dati gestiti. Risulta dunque necessario trovare un'alternativa che soddisfi le necessità attuali e future, in ambito di scalabilità e ottimizzazione.

### 1.3 Obiettivi di Tesi

L'obiettivo della Tesi risulta essere quello di realizzare un prototipo di FS che sia scalabile, in grado di effettuare rapide letture e scritture, strutturalmente solido e ottimizzato nell'allocazione di spazio.

In particolare, il lavoro si concentra sulla progettazione e realizzazione di un prototipo alternativo al sistema preesistente, con l'intento non solo di mantenere

le prestazioni attuali, ma, ove possibile, di migliorarle.

### 1.3.1 Domande

- Quali tipi di dati devono essere memorizzati e gestiti?
- Qual è la quantità stimata di dati da gestire a regime?
- Quali saranno le dimensioni massime dei file e la granularità delle operazioni sui dati?
- Come garantire un basso consumo di risorse?
- Qual è la struttura di archiviazione più adatta?
- Il FS dovrà scalare per gestire un numero crescente di dispositivi IoT?
- Qual è la strategia per l'espansione dello spazio di archiviazione?
- Come gestire guasti hardware o interruzioni improvvise di alimentazione?
- È necessario prevedere meccanismi di backup o snapshot per i dati?
- Come si possono introdurre nuove funzionalità senza compromettere i dati esistenti?

### 1.3.2 Scopo

L'obiettivo finale è quello di apportare un significativo miglioramento al software esistente, sfruttando l'hardware già disponibile per ottenere prestazioni più elevate. Questo intervento mira a incrementare le capacità del prodotto, garantendo una maggiore efficienza e un'esperienza d'uso più soddisfacente per i clienti.

In particolare, il miglioramento si traduce nell'ottimizzazione delle funzionalità attuali e nell'ampliamento delle possibilità offerte dal sistema, lavorando direttamente sul software. Questo approccio consente di massimizzare il valore del prodotto, andando incontro alle crescenti aspettative degli utenti e mantenendo un vantaggio competitivo sul mercato.

## 1.4 Struttura della Tesi

Questa Tesi è strutturata in tre capitoli principali.

Il primo capitolo fornisce le basi teoriche del lavoro, offrendo una panoramica sui moderni FS e sulla loro connessione con il mondo dell'Internet of Thing (IoT). Viene inoltre approfondita la struttura dati B-Tree, evidenziandone le proprietà principali e la sua rilevanza rispetto al problema affrontato.

Il secondo capitolo si concentra sulla progettazione e sull'implementazione del prototipo di FS. Vengono descritte le scelte effettuate durante il processo di sviluppo e spiegato come il B-Tree sia stato utilizzato per soddisfare le esigenze specifiche della piattaforma IOtto.

Infine, il terzo capitolo discute i risultati ottenuti dalla valutazione del prototipo. Vengono inoltre presentate le conclusioni tratte dallo studio e delineate le possibili direzioni per future modifiche e sviluppi relativi al prodotto.





---

# Chapter 2

## State of the art

### 2.1 L'Internet of Things

Il concetto di Internet of Things, o in italiano Internet delle Cose, rappresenta un'evoluzione nell'utilizzo della rete Internet, in cui gli oggetti, o "cose", diventano riconoscibili e acquisiscono la capacità di comunicare dati attraverso la rete e di accedere a informazioni aggregate da altre fonti.

Uno dei primi utilizzi di questo termine risale al 2001, in un documento del centro Auto-ID relativo alla creazione del network EPC, concepito per tracciare automaticamente il flusso di beni nella catena di fornitura.

Tuttavia, non esiste un'origine univoca o un creatore effettivo del termine, poiché il concetto si è sviluppato progressivamente attraverso contributi di diversi ricercatori e innovatori nell'arco di tempo che va dai primi anni '90 fino al 2010.

Gli oggetti connessi, che costituiscono il cuore pulsante dell'IoT, sono definiti "oggetti intelligenti" (Smart Objects) e si caratterizzano per proprietà distintive come la capacità di identificarsi, connettersi, localizzarsi, elaborare dati e interagire con l'ambiente circostante.

Attraverso tecnologie come le etichette RFID (Identificazione a radiofrequenza) o i codici QR, questi oggetti e luoghi possono comunicare informazioni tramite la rete o dispositivi mobili.

L'obiettivo principale dell'IoT risulta essere dunque il tentativo di creare una mappatura elettronica del mondo fisico, attribuendo un'identità digitale agli oggetti e ai luoghi che lo compongono.

Sono nati inoltre organizzazioni come EPCglobal e Open Geospatial Consortium (o OGC) che lavorano per creare standard globali per la visibilità dei dati e l'utilizzo di sensori connessi via Web.

In conclusione, le applicazioni dell'IoT sono molteplici e variano dai processi industriali alla logistica, dalla logistica all'efficienza energetica, fino all'assistenza remota e alla salvaguardia ambientale.

[Uck11][HBE11][Cha13]

## 2.2 File System

Tutti i processi e le applicazioni informatiche necessitano di archiviare e recuperare informazioni, ed è possibile, durante la loro esecuzione di memorizzare una quantità limitata di informazioni all'interno del proprio spazio di indirizzamento.

Questo metodo di gestione dei dati però non viene sprovvisto di problematiche:

- Capacità limitata, in quanto lo spazio di indirizzamento virtuale limita la quantità di dati archiviabili, rendendolo inadeguato per applicazioni che richiedono un'ampia archiviazione di dati, come sistemi bancari o database.
- Le informazioni memorizzate nello spazio di indirizzamento di un processo sono altamente volatili e vengono perse al termine del processo stesso.
- Non è permesso l'accesso concorrente. Spesso è necessario che più processi possano accedere contemporaneamente alle stesse informazioni, ma il confinamento dei dati all'interno di un unico processo lo impedisce.

Per rispondere a queste sfide, l'archiviazione a lungo termine richiede il rispetto di tre requisiti fondamentali:

- La possibilità di memorizzare grandi quantità di informazioni.

- La persistenza delle informazioni, anche dopo la terminazione del processo.
- L'accesso simultaneo ai dati da parte di più processi.

Per affrontare queste problematiche, è stato sviluppato il concetto di File System, una componente fondamentale dei sistemi informatici progettata per permettere la gestione completa dei dati, inclusa la loro creazione, modifica, archiviazione e recupero all'interno di un sistema.

Un file system, al suo livello più alto, fornisce dunque un modo organizzato per archiviare, recuperare e gestire informazioni su supporti di archiviazione permanenti come i dischi.

Esistono diversi approcci alla gestione dell'archiviazione permanente, poiché il problema è ampio e offre numerose soluzioni possibili. [Gia98][TB15]

## 2.3 Categorizzazione dei File System

I file system possono essere suddivisi in diverse categorie in base a specifiche caratteristiche, tra cui il livello di organizzazione gerarchica. Di seguito, analizziamo due tipologie principali di file system in relazione alla loro struttura gerarchica.

### 2.3.1 Single-Level Directory Systems

Questo tipo di file system rappresenta la forma più basilare e semplice di organizzazione dei dati. Consiste in una singola directory, spesso denominata root, che contiene tutti i file del sistema.

Non esistono ulteriori livelli o sottodirectory: tutti i file sono archiviati in un unico spazio condiviso.

L'approccio a livello unico offre alcuni vantaggi distinti. In primo luogo, la semplicità della struttura lo rende facile da implementare e utilizzare, poiché non sono necessari meccanismi complessi per navigare tra directory o sottodirectory. Inoltre, la localizzazione dei file è immediata, poiché tutto è contenuto all'interno di una singola cartella, rendendo inutile un sistema di ricerca avanzato.

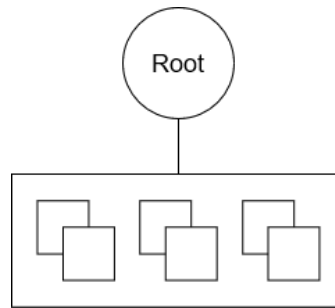


Figure 2.1: Esempio di Single Level Directory System

Tuttavia, questo modello ha limitazioni significative, soprattutto per applicazioni più complesse o ambienti che richiedono la gestione di un elevato numero di file.

Per questo motivo, i sistemi a directory singola sono utilizzati principalmente in dispositivi embedded o sistemi con esigenze limitate, come fotocamere digitali o dispositivi elettronici di consumo basilari. In tali contesti, la semplicità e la leggerezza dell'architettura superano le necessità di una maggiore organizzazione.

### 2.3.2 Hierarchical Directory Systems

A differenza del modello a singolo livello, i sistemi a directory gerarchica sono progettati per rispondere alle esigenze dei moderni ambienti informatici, caratterizzati dalla gestione di grandi volumi di dati e file.

In questo tipo di file system, è possibile creare una struttura organizzata composta da directory e sottodirectory. Questa struttura consente di suddividere i file in categorie o gruppi logici, favorendo una gestione più efficiente e un accesso più rapido alle informazioni.

L'organizzazione gerarchica risulta particolarmente utile quando il sistema deve gestire elevate quantità di file. La possibilità di classificare i file in directory dedicate consente di mantenere ordine e chiarezza, riducendo la complessità che deriverebbe dall'avere tutto in un'unica directory.

Un ulteriore vantaggio di questo approccio emerge nei contesti multiutente, dove più utenti condividono lo stesso file server. In questi casi, ogni utente può

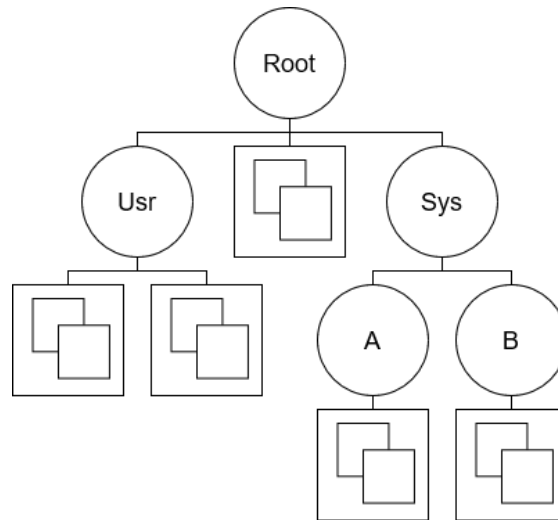


Figure 2.2: Esempio di Hierarchical Directory System

disporre di una directory root personale, che funge da punto di partenza per una gerarchia dedicata. Questa configurazione garantisce non solo la separazione dei dati tra gli utenti, ma anche un livello di personalizzazione che si adatta alle esigenze individuali.

In conclusione, mentre i sistemi a livello singolo sono ideali per applicazioni semplici e specifiche, i sistemi gerarchici rappresentano una soluzione robusta e scalabile per ambienti complessi e moderni.

[TB15]

## 2.4 Data Structures

Una struttura dati è un modo per memorizzare e organizzare i dati al fine di facilitare l'accesso e le modifiche.

Sono elementi fondamentali nella progettazione e nell'implementazione dei file system, poiché determinano come i dati vengono organizzati, memorizzati e recuperati, influenzando direttamente l'efficienza e le prestazioni delle operazioni sui dati.

Nessuna struttura dati singola è ottimale per tutti gli scopi, quindi è importante

conoscere i punti di forza e le limitazioni delle diverse strutture.

La scelta della struttura dati appropriata dipende da vari fattori, tra cui le esigenze specifiche del sistema, le prestazioni richieste e le limitazioni hardware.

Tra le strutture dati più comuni utilizzate nei file system, si possono citare:

- **Linked List:** una sequenza di nodi collegati tra loro, in cui ogni nodo contiene un riferimento al nodo successivo.

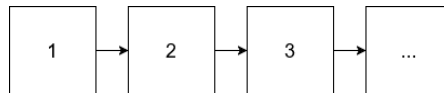


Figure 2.3: Esempio di Linked List

- **Array:** una collezione di elementi disposti in una sequenza lineare, in cui ogni elemento è accessibile tramite un indice numerico.

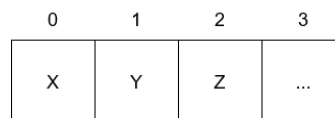


Figure 2.4: Esempio di Array

- **Tree:** una struttura gerarchica composta da nodi collegati tra loro, in cui ogni nodo può avere uno o più nodi figli.

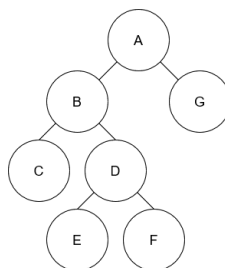


Figure 2.5: Esempio di Tree

- **Hash Table:** una struttura dati che associa chiavi a valori, consentendo un accesso rapido e diretto ai dati.

[CLRS22]

Key	Value
A	1
B	2
C	3
...	...

Figure 2.6: Esempio di Hash Table

## 2.5 Strutture Dati Basate sui Tree

Le strutture dati come Linked List e Array sono strumenti potenti per rappresentare relazioni lineari, dove gli elementi sono disposti in una sequenza. Tuttavia, non tutte le relazioni nel mondo reale sono lineari.

Molti problemi richiedono la rappresentazione di relazioni gerarchiche o ramificate, dove un elemento può essere connesso a più elementi in modo non sequenziale.

Un Tree è una struttura dati non lineare composta da nodi collegati tra loro in modo gerarchico.

Ogni albero ha un Nodo Radice denominato Root da cui partono uno o più Nodi denominati Figli, e ogni Figlio può a sua volta avere altri Figli, formando una struttura ramificata.

In questo capitolo esploreremo le principali tipologie di alberi, le loro caratteristiche, i vantaggi e gli svantaggi.

### 2.5.1 Binary Trees

Un Binary Tree è la variante più basilare dei Tree in cui ogni nodo può avere al massimo due Figli, chiamati Figlio Sinistro e Figlio Destro, ed è rappresentato tramite tre attributi per nodo: ‘Key’ (Il Valore), ‘Left’ (Figlio Sinistro) e ‘Right’

(Figlio Destro).

Questa struttura è semplice, versatile e ampiamente utilizzata per realizzare strutture dati più avanzate.

Un Binary Tree risulta essere semplice, efficiente e flessibile, ma può risultare inefficiente in caso di sbilanciamento e non è adatto a rappresentare relazioni con più di due figli.

### 2.5.2 Binary Search Trees

Un Binary Search Tree è un tipo di Binary Tree in cui ogni nodo rispetta un ordine: il valore del nodo è maggiore di tutti i valori nel suo sottoalbero sinistro e minore di tutti quelli nel sottoalbero destro.

Questa struttura è utile per operazioni efficienti di ricerca, inserimento e cancellazione, e mantiene i nodi ordinati, semplificando operazioni come la ricerca del minimo o del massimo.

Tuttavia, se l'albero non è bilanciato, può degenerare in una lista collegata perdendo così tutti i suoi benefici.

### 2.5.3 Balanced Trees

I Balanced Trees, come AVL e Red-Black, sono Binary Search Trees che mantengono l'altezza logaritmica rispetto al numero di nodi per garantire efficienza.

Sia gli AVL che i Red-Black Trees utilizzano algoritmi e regole specifiche per mantenere l'equilibrio.

Entrambi offrono prestazioni garantite come ricerca, inserimento e cancellazione in tempi brevi, rendendoli ideali per applicazioni che richiedono velocità e prevedibilità.



Tuttavia, la gestione del bilanciamento richiede algoritmi più complessi rispetto ai BST standard, con un maggiore overhead computazionale dovuto alle operazioni di bilanciamento.

### 2.5.4 N-ary Trees

Un N-ary Tree è una generalizzazione dei Binary Trees in cui ogni nodo può avere fino a N Figli, con N come costante.

Può essere rappresentato tramite array o liste di puntatori ai figli ed è utile per relazioni con un numero fisso di figli, come negli alberi di decisione nei giochi di scacchi.

Il problema principale con gli N-ary Tree è la scelta del valore di N.

Se il valore è troppo piccolo, le operazioni risultano efficienti, ma può verificarsi uno spreco di memoria quando molti nodi hanno meno di N figli, e la struttura risulta rigida per relazioni con un numero variabile di figli.

Mentre se il valore è troppo grande, la struttura tende a diventare troppo simile ad una Linked List, con un aumento della complessità computazionale.

[CLRS22]

## 2.6 B-Tree

Un B-Tree è un Balanced Search Tree progettato per mantenere i dati ordinati e permettere operazioni di ricerca, inserimento e cancellazione in tempo logaritmico.

A differenza di altri Balanced Trees come i Red-Black Trees, i B-Tree sono ottimizzati per ridurre al minimo il numero di accessi al disco, rendendoli ideali per applicazioni che gestiscono grandi quantità di dati su dispositivi di memorizzazione secondaria.

Indicando con  $n$  il numero massimo di Nodi Figli a cui un Nodo può puntare, un B-Tree ha le seguenti proprietà:

- Il numero massimo di Chiavi per Nodo è  $n - 1$ .

- Il Nodo Root è o un Nodo Foglia o un Nodo Interno con un numero di Figli compreso tra 2 e  $n$  e un numero di chiavi compreso tra 1 e  $n - 1$ .
- Tutti gli altri Nodi Interni hanno un numero di Figli compreso tra  $n/2$  e  $n$ .
- Tutte le Foglie si trovano alla stessa profondità.

Queste proprietà garantiscono che il B-Tree rimanga bilanciato, mantenendo un'altezza relativamente bassa anche con un grande numero di elementi.

---

## Chapter 3

# Design and Implementation

### 3.1 Progettazione

Riprendendo l'elenco di domande poste precedentemente in section 1.3.1 come base di partenza, la progettazione si è basata sul cercare di trovare una soluzione a tutte le domande elencate.

La prima domanda che ci siamo trovati ad affrontare è stata quella relativa ai tipi di dati da memorizzare e gestire, che ci ha portato davanti al problema dell'Eterogeneità dei Dati.

L'ecosistema dell'IoT è caratterizzato dalla presenza di numerosi organismi di standardizzazione, ciascuno dei quali contribuisce con un ampio ventaglio di norme e protocolli che spesso si sovrappongono o risultano in conflitto tra loro.

Questa diversità si riflette nei dati acquisiti dai dispositivi IoT, che vengono prodotti in una combinazione di formati, inclusi dati strutturati, semi-strutturati e non strutturati, generando ulteriori complessità.

L'eterogeneità nei formati e nei protocolli non si limitano alla fase di acquisizione, ma hanno un impatto diretto anche nelle fasi di archiviazione, complicando la gestione dei dati nei sistemi di storage.

Per ovviare a ciò nella piattaforma veniva usato un sistema di storage basato su file system tradizionali, che non era in grado di gestire in modo efficiente la varietà di dati prodotti dai dispositivi IoT.

### 3.1.1 Requisiti

- Il FS deve essere in grado di gestire grandi quantità di dati.
- Il FS deve garantire un accesso rapido ai dati.
- Il FS deve essere strutturalmente solido e affidabile.
- Il FS deve essere ottimizzato per l'allocazione spazio.
- Il FS deve essere scalabile per gestire un numero crescente di dispositivi IoT.
- Il FS deve prevedere meccanismi di backup e snapshot per i dati.
- Il FS deve garantire la coerenza dei dati in caso di guasti hardware o interruzioni di alimentazione.

### 3.1.2 Architettura

Il prototipo di FS è basato sulla struttura dati B-Tree, progettato per garantire un'efficienza ottimale nella gestione dei dati.

La struttura del B-Tree consente di mantenere i dati ordinati e di effettuare operazioni di ricerca, inserimento e cancellazione in tempo logaritmico, garantendo prestazioni elevate e una gestione efficiente delle risorse.

Il prototipo è composto da tre componenti principali:

- **B-Tree:** la struttura dati principale che gestisce l'organizzazione e la memorizzazione dei dati.
- **File System Interface:** l'interfaccia che consente alle applicazioni di interagire con il FS per accedere, modificare e gestire i dati.
- **Storage Layer:** il livello di archiviazione che si occupa di memorizzare fisicamente i dati su disco.

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         // Prints "Hello, World" to the terminal window.  
4         System.out.println("Hello, World");  
5     }  
6 }
```

La progettazione del prototipo si basa su un approccio modulare e scalabile, che consente di estendere e adattare il sistema alle esigenze

You may also put some code snippet (which is NOT float by default), eg: section 3.1.2.

## 3.2 Fancy formulas here



---

# Bibliography

- [Cha13] Hakima Chaouchi. *The internet of things: Connecting objects to the web*. John Wiley & Sons, 2013.
- [CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.
- [Gia98] Dominic Giampaolo. *Practical file system design with the Be file system*. Morgan Kaufmann Publishers Inc., 1998.
- [HBE11] Olivier Hersent, David Boswarthick, and Omar Elloumi. *The internet of things: Key applications and protocols*. John Wiley & Sons, 2011.
- [TB15] Andrew S Tanenbaum and Herbert Bos. *Modern operating systems*. Pearson Education, Inc., 2015.
- [Uck11] D Uckelmann. *Architecting the Internet of Things*. Springer, 2011.

## BIBLIOGRAPHY

---



---

# Acknowledgements

Optional. Max 1 page.