

ALMA MATER STUDIORUM • UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

**Lightly: un modello per stack undo-redo leggeri per
l'editing cooperativo**

Relatore: Chiar.mo

Prof. Fabio Vitali

Presentata da:

Francesco Fornari

III Sessione

Anno Accademico 2020-2021

Indice

Indice	2
Indice delle illustrazioni	4
1. Introduzione	6
2. Contesto scientifico e tecnologico	11
2.1 Storia dei Word Processor	11
2.2 Operational Transformation e Versioning	13
2.2.1 Operational Transformation	14
2.2.2 Versioning	15
3. Lightly	17
3.1 Il progetto Lightly	17
3.2 I tre livelli di modifica	17
3.3 Modifica Meccanica	18
3.4 Modifica Strutturale	19
3.5 Annullamento della modifica	21
3.6 Sessione di esempio	22
4. Architettura di lightly	28
4.1 Introduzione	28
4.2 Albero	30
4.2.1 Nodi	30
4.2.2 Contenuto dell'albero	30
4.3 Struttura delle modifiche	31
4.3.1 Modifica meccanica	31
4.3.2 Modifica strutturale	32
4.3.3 Stack UNDO-REDO	33
4.3.4 Esecuzione di UNDO e REDO	34
4.4 Range	35
4.4.1 Posizione del cursore	35
4.4.2 Intervallo del range	36
4.5 Modifica	37
4.5.1 Rilevazione del cambiamento	37
4.5.2 Cattura della modifica	38
4.5.3 Valutazione della modifica	39
4.5.4 Salvataggio della modifica	40
4.5.5 Update delle modifiche	40


5. Valutazione	42
5.1 Primo test	42
5.2 Secondo test	43
5.3 Terzo test	45
5.4 Conclusioni del test	47
6. Conclusione	48
7. Bibliografia	50

Indice delle illustrazioni

1. Primi design del NES per il Nord America	12
2. Esempio di Operational Transformation su Google Document	14
3. Struttura ad albero a solo foglie delle versioni	15
4. Struttura lineare delle versioni	15
5. Struttura ad albero delle versioni	16
6. Contenuto Modifica Meccanica	18
7. Contenuto Modifica Strutturale	20
8. Tabella dei casi di UNDO-REDO	21
9. Lightly: Inizio	22
10. Lightly: Inserimento testo	22
11. Lightly: Inserimento testo	23
12. Lightly: Correzione errore	23
13. Lightly: Fine frase	24
14. Lightly: Bold e Italic	25
15. Lightly: Unwrap	26
16. Lightly: Undo	26
17. Lightly: Ulteriori Undo	27
18. Struttura ad albero di TinyMCE	30
19. Stato di TinyMCE	31
20. Modifica Meccanica	31
21. Modifica Strutturale	32
22. Modifiche Meccaniche all'interno delle Modifiche Strutturali	33
23. Pila delle modifiche	33
24. Verso di Undo e Redo	34
25. I due tipi di Modifica Meccanica	34
26. Prima e dopo UNDO	35
27. Posizione del puntatore	35
28. Range sul documento	36

29. Range dopo un invio	37
30. Controllo dell stringa	38
31. Regole per la determinazione ei tipi strutturali	40
32. Compatibilità degli update delle modifiche	41
33. Primo test con lightly	42
34. Primo test senza lightly	43
35. Grafico primo test	43
36. Secondo test con lightly	44
37. Secondo test senza lightly	44
38. Grafico secondo test	45
39. Terzo test con e senza lightly	46
40. Grafico terzo test	46
41. Grafico risorse utilizzate durante gli UNDO con lightly	47
42. Grafico risorse utilizzate durante gli UNDO con TinyMCE	47

1. Introduzione

Un **Word Processor** è un programma che permette di scrivere e modificare file di testo complessi su un computer, permette di aggiungere cambiamenti grafici sul testo come il **Grassetto**, il Sottolineato, il *Corsivo*, cambiare **Colore**, modificare il **font** dei caratteri, 6. creare elenchi, aggiungere [link](#) a siti web o aggiungere immagini e loghi .

I primi sistemi di questo tipo risalgono agli anni '60 quando vennero diffuse le macchine da scrivere automatiche di IBM. Negli anni '70 il mercato fu conquistato da minicomputer specializzati per uffici ed aziende. Successivamente negli anni '80 e '90 la loro diffusione aumentò, con l'arrivo nel 1983 con l'avvento di Microsoft Word per MS-DOS, probabilmente il più famoso Word Processor al mondo. Fonte: [TH06]

Questi software hanno permesso alle persone di poter scrivere, vedere istantaneamente il risultato del proprio lavoro e cosa più importante dare la possibilità modificare parti di testo già scritto. Su una macchina da scrivere tradizionale era impossibile fare questa operazione, ed è questa tra le tante ragioni che hanno permesso ai Word Processor di sostituire una macchina da scrivere.

Con l'avanzamento delle prestazioni dei computer e la diffusione di internet si è iniziato a portare i Word Processor al di fuori di una tradizionale applicazione per computer, portandoli sui siti internet.

Questo particolare tipo di Word Processor viene chiamato Web Word Processor (WWP). Questi strumenti permettono di creare documenti esattamente come i Word Processor tradizionali ma a differenza di questi sono utilizzabili attraverso un Browser Web. Esistono svariati Word Web Processor come Microsoft Word 365, Google Document, **TinyMCE**.

In particolare TinyMCE ha un difetto molto grave, il consumo eccessivo di **memoria primaria** o **RAM**.

La RAM è la memoria volatile ad alta velocità di utilizzo e viene impiegata dai programmi per tenere in memoria i dati ed il programma stesso. Avere una memoria rapida permette di dare ai programmi in esecuzione la sensazione di essere in tempo reale.

La nostra teoria è che TinyMCE utilizza molta RAM per via dell'eccessivo spazio occupato dall'UNDO-REDO Stack.

Un UNDO-REDO Stack è la struttura dati che contiene lo storico delle modifiche eseguite sul documento di testo. Queste modifiche salvate sono in grado di essere lette dal Word Processor ed essere utilizzate per andare a ritroso nelle modifiche eseguite sul documento.

Il caso in cui l'utente voglia annullare la modifica eseguita viene definito UNDO, mentre il caso in cui l'utente voglia riapplicare la modifica annullata viene definito REDO.

Per un qualsiasi Word Processor è fondamentale avere un UNDO-REDO Stack, perché è una delle funzioni più utili per l'utente.

L'UNDO-REDO Stack è gestito dal Change Tracking. Il sistema di Change Tracking si occupa di intercettare la modifica eseguita, incapsularla e salvarla sull'UNDO-REDO Stack. Esistono più strutture per salvare la modifica e vedremo una di queste più avanti, nel corso della tesi.

TinyMCE utilizza una tecnica di Change Tracking ed una gestione degli avvenimenti UNDO-REDO non ben ottimizzate. Per questo motivo il sistema spreca una quantità eccessiva di risorse, a tal punto che si eseguono delle modifiche su documenti pesanti il sistema non risponde più ai comandi dell'utente.

TinyMCE è una libreria JavaScript. Permette facilmente di implementare all'interno di pagine internet un Web Word Processor con le caratteristiche di un classico Word Processor.

JavaScript è un linguaggio di programmazione principale per la logica dei siti web lato front-end, per questo motivo è molto diffuso e si trovano facilmente esempi e documentazione per utilizzarlo al meglio. TinyMCE offre una ben documentata libreria di API, Application Program Interface. Le API permettono di interfacciarsi con il programma per ottenere dati o modificare l'esecuzione dello stesso.

Avere a disposizione delle API ben documentate rende più facile creare plugin. Un plugin è un programma distinto dal programma principale, ma non autonomo. Il plugin permette di aggiungere delle funzionalità al programma principale non presenti in origine.

La mia ipotesi è che utilizzare un sistema di Change Tracking e uno Stack UNDO-REDO basato sull'articolo di Di Iorio, Spinaci e Vitali - Multi-layered edits for meaningful interpretation of textual differences, che sia più leggero sulla RAM e che permetta a TinyMCE di essere utilizzato anche per l'editing di documenti più pesanti in maniera fruibile.

Nell'articolo citato [DSV19] si utilizzano vari livelli di astrazione per definire una modifica ed il suo tipo.

I dati necessari per il primo livello di astrazione sono:

- Tipo di modifica
- Porzione di testo modificata
- Posizione della modifica nel testo
- ID
- Creatore della modifica
- Data della modifica

Nel livello di astrazione più basso la modifica può essere di due tipi, di inserimento di contenuto definito INS o di cancellazione di contenuto definito DEL. Queste sono le due operazioni più semplici effettuabili su un documento. Si possono usare queste due operazioni per descrivere modifiche più complesse. Le modifiche di tipo INS e DEL le chiameremo Modifiche Meccaniche, queste stanno al primo livello di modifica chiamato livello delle modifiche meccaniche.

Le singole operazioni meccaniche possono essere aggregate in una struttura più complessa che d'ora in poi definiremo Modifica Strutturale.

Le modifiche strutturali permettono di esprimere meglio il tipo di modifica eseguita, differenziando le modifiche in due categorie, quelle basate sul contenuto del documento e quelle basate sullo stile del documento.

Le modifiche strutturali contengono queste informazioni:

- ID
- Tipo di modifica
- Creatore della modifica
- Data della modifica
- Lista delle modifiche meccaniche associate
- Posizione del cursore prima della modifica
- Posizione del cursore dopo la modifica

Le modifiche strutturali permettono di avere un Change Tracking decisamente più leggibile rispetto alle modifiche meccaniche e utilizza una quantità di spazio minima. Le modifiche strutturali definiscono il secondo livello di astrazione chiamato livello delle modifiche strutturali.

Sopra il livello le modifiche strutturali viene definito il livello delle **modifiche semantiche**. Questo tipo di modifiche esprimono il significato semantico della modifica, rendendola più comprensibile per un essere umano.

Avere un buon sistema di gestione di modifiche del documento permette di essere utilizzato per features aggiuntive di Word Processor, come il Versioning e l'Operational Transformation.

Il Versioning è un sistema di versionamento di file, esprime la storia del documento e delle sue versioni. Per esprimere il cambiamento da una versione ad un'altra si utilizzano insiemi di modifiche.

L'Operational Transformation è un sistema di scrittura collaborativa, permette a più utenti di lavorare contemporaneamente allo stesso documento da più dispositivi. I dispositivi per lavorare in contemporanea si scambiano continuamente le modifiche.

Per dimostrare la mia tesi ho creato lightly. Lightly è un plugin per TinyMCE che introduce un Change Tracking ed un Undo-Redo Stack basato sull'articolo [DSV19]. Inoltre lightly offre una buona base di partenza per sviluppare progetti di Versioning ed Operational Transformation.

Ogni volta che viene scatenato un evento che modifica il documento, come la pressione di un tasto oppure l'aggiunta di grassetto, il programma effettua una ricerca di tipo testuale sul documento, trova la modifica e la salva secondo la struttura descritta precedentemente.

Per valutare questa mia tesi ed il mio plugin sono state create due pagine web con due versioni di TinyMCE, una con Lightly e l'altra senza. In seguito sono stati eseguiti dei test identici su documenti molto pesanti, i test consistono nell'eseguire le stesse operazioni su un documento, come la creazione di nuove parti sul documento, la trasformazione dello stile del documento e l'annullamento delle modifiche effettuate.

Questi test sono valutati su due parametri: l'utilizzo di RAM e la percezione della velocità del programma a rispondere su documenti complessi.

Una buona acquisizione delle modifiche deve permettere di superare i test descritti sopra e di avere un sistema leggero di salvataggio di modifiche. Avere un sistema di questo tipo consente di utilizzare questa stessa struttura per il Versioning o l'Operational Transformation.

In particolare l'Operational Transformation essendo basato sulla connessione e scambio di dati di più terminali attraverso la rete lo rende perfetto per un Web Word Processor. Questo perché non richiede ai suoi utilizzatori di avere lo stesso programma installato sul computer ma necessita soltanto un Web Browser connesso alla stessa pagina web. Inoltre per condividere il documento basta l'invio di un link web, rendendo il tutto più facile da utilizzare per un utente.

2. Contesto scientifico e tecnologico

2.1 Storia dei Word Processor

Negli anni i Word Processor hanno sostituito le vecchie macchine da scrivere in quanto consentono di modificare parti di testo già scritte.

Dagli '80 i Word Processor sono entrati nella vita quotidiana di qualsiasi impiegato d'ufficio che ha la necessità di scrivere testi.

Negli anni '90 grazie all'abbassamento del costo dei computer, all'avvento di stampanti ad alta qualità e all'avanzamento tecnologico i Word Processor diventano di uso comune per qualsiasi persona che voleva scrivere testi.

Successivamente con l'avvento della posta elettronica e di internet i Word Processor diventano fondamentali anche per inviare documenti e testi sostituendo i fax.

Negli anni '70 IBM ha iniziato ad approcciarsi al mondo dei Word Processor. In principio il Word Processor non era stato pensato come un programma per scrivere documenti, ma era inteso come un sistema di macchine da scrivere automatizzate, usate per centralizzare la scrittura e la trascrizione di documenti. Sostanzialmente era concepito come metodo di "Data Processing", e si basava sull'elaborazione di informazioni e dati per le grandi aziende, i Word Processing Center.

Queste macchine da scrivere automatizzate e macchine per la dettatura erano chiamate Word Processor Machines. Al tempo il concetto di Word Processor non era inteso come un programma per la scrittura di documenti ma come una macchina da scrivere automatizzata.

Il Word Processor non è nato come una invenzione, ma come una evoluzione di strumenti già esistenti, trasformati ed impacchettati per soddisfare le esigenze del mercato, offrendo strumenti creati appositamente per la produzione di documenti.

Le caratteristiche principali di un Word Processor sono state ereditate dai Text Editor, esistenti già negli anni '60, usati per la creazione e manipolazione di codice per i programmatori.

Negli anni '70 il concetto di Word Processor passa da una macchina da scrivere automatizzata a minicomputer, concepiti esclusivamente per creare e modificare documenti. Solo più tardi questo termine è stato associato ad un programma per computer.

Da questa decade, in coincidenza con la creazione del mercato di strumenti per la videoscrittura di documenti, ha iniziato a prendere piede il termine Word Processing. Il termine è stato coniato per la prima volta a metà anni '50 da Ulrich Steinhilber, un Sales Executive Tedesco di macchine da scrivere IBM.

Alla fine degli anni '70 e durante gli anni '80 il concetto di Word Processor diventa quello che anche noi conosciamo, passando da minicomputer dedicati a dei programmi per computer concepiti per utilizzare più applicazioni.

Programmi come EasyWriter, WordStar e MultiMate sono diventati sempre più delle ottime imitazioni per i più sofisticati e costosi minicomputer dedicati.



Fig. 1 - Primi design del NES per il Nord America

Negli anni '80 e '90 l'esigenza di avere un sistema economico di videoscrittura è diventata sempre più importante. Basta pensare a come la Nintendo, per essere più appetibile nel mercato statunitense, voleva vendere il NES come un sistema sia di videogioco che di videoscrittura.

Negli anni '80 i Word Processor diventano sempre più avanzati ed escono alcuni dei più importanti prodotti in uso ancora oggi, nell'83 Microsoft Word per il Sistema Operativo DOS e nell'84 AppleWorks per i computer Apple IIe, che si evolverà in seguito in Pages per i sistemi Mac attuali.

In questo periodo iniziano ad essere introdotti concetti fondamentali per i Word Processor come WYSIWYG, What You See Is What You Get. Questo consente nel visualizzare sullo schermo esattamente quello che verrà stampato, conservando la stessa impaginatura e le stesse caratteristiche grafiche prodotte sullo schermo.

Per via della scarsa velocità di internet, prima degli anni 2000, praticamente si era ancora costretti a stampare il documento per condividere la versione cartacea, infatti, se si voleva inviare un documento ad un'altra persona era molto più veloce stamparlo ed inviarlo via fax, e quando arrivava all'altro utente per salvarlo su un altro computer lo si scannerizza, ottenendo una versione non modificabile. Ora con il miglioramento della velocità internet e con il calo del costo della tecnologia si è passati a inviare file attraverso la rete, evitando di stampare e sprecare carta.

Dopo gli anni 2000 con il progresso delle tecnologie e dei browser web sono stati sviluppati programmi per il lavoro d'ufficio online, tra questi strumenti sono presenti anche i Web Word Processor, i più diffusi sono Microsoft Word 365 e Google Documents.

Questi software, oltre ad offrire prodotti senza la necessità di installare altri programmi ad eccezione del browser web, permettono anche di condividere documenti, lavorare in più persone semplicemente scambiandosi link e soprattutto di essere utilizzabili su qualsiasi dispositivo connesso ad internet.

2.2 Operational Transformation e Versioning

L'Operational Transformation ed il Versioning sono due concetti per i Word Processor utili per migliorare il lavoro cooperativo tra due o più persone.

Queste due features sono sempre più richieste dalle aziende, in quanto permettono di velocizzare il lavoro di gruppo. Entrambe utilizzano le modifiche sul documento come base su cui partire per funzionare ed esprimersi.

2.2.1 Operational Transformation

L'Operational transformation è il sistema che permette di lavorare in due o più persone contemporaneamente alla scrittura di documenti.

Il cuore dell'Operational Transformation è la gestione ed unione delle modifiche concorrenti sullo stesso documento.

I cambiamenti sul documento vengono generati da più dispositivi contemporaneamente solitamente connessi attraverso internet.

Un Web Word Processor per questo tipo di applicazioni è molto comodo, perchè permette di esportare la stessa applicazione, sempre aggiornata all'ultima versione, su un qualsiasi dispositivo connesso alla rete.

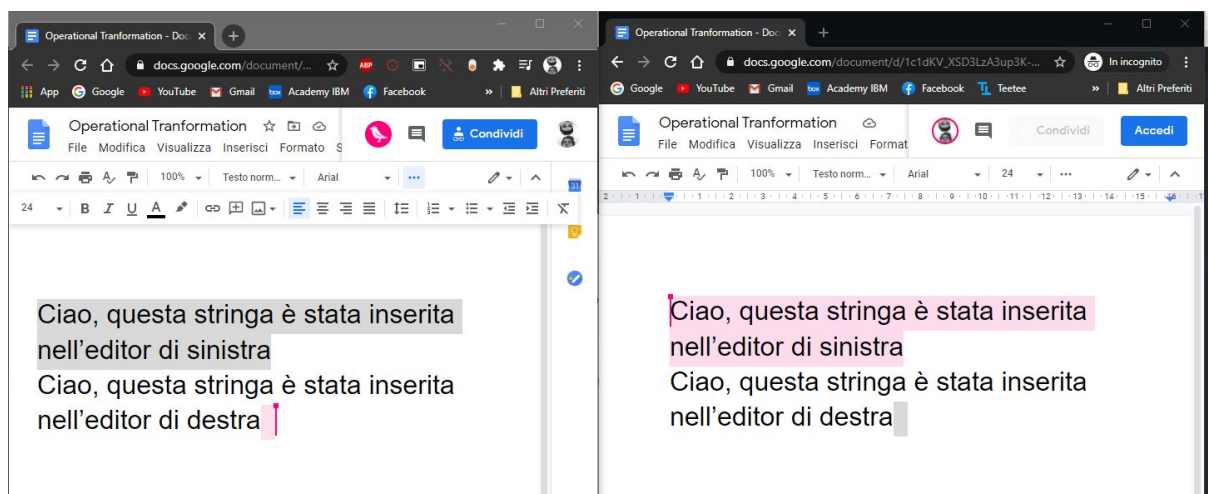


Fig. 2 - Esempio di Operational Transformation su Google Document

Questa è una schermata di esempio dell'Operational Transformation su Google Document, sono 2 interfacce web connesse allo stesso documento che possono effettuare modifiche contemporaneamente.

Avere a disposizione un sistema di Change Tracking efficiente ed un Undo-Redo che consuma poco spazio in memoria è fondamentale per scambiarsi le modifiche in maniera rapida e che non sia causa di conflitti.

2.2.2 Versioning

Il Versioning è un sistema di gestione di uno stesso documento in più versioni, può studiare e generare documenti in autonomia.

Possono esistere vari percorsi per avere più versioni di un documento.

Si potrebbe partire da una stessa base e specializzare ogni versione, ad esempio della documentazione che può variare in base al tipo di utente interessato.

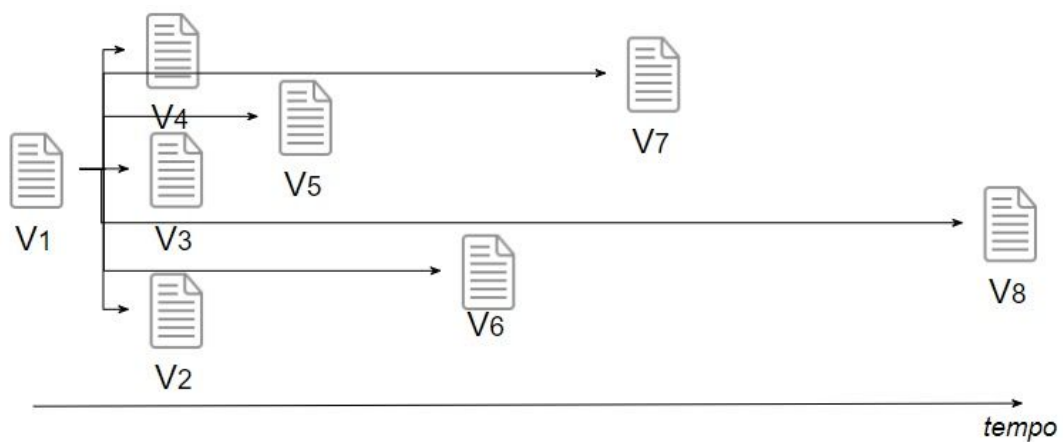


Fig. 3 - Struttura ad albero a solo foglie delle versioni

Oppure ci potrebbe essere un percorso di aggiornamento nel tempo dello stesso documento, ad esempio un curriculum.

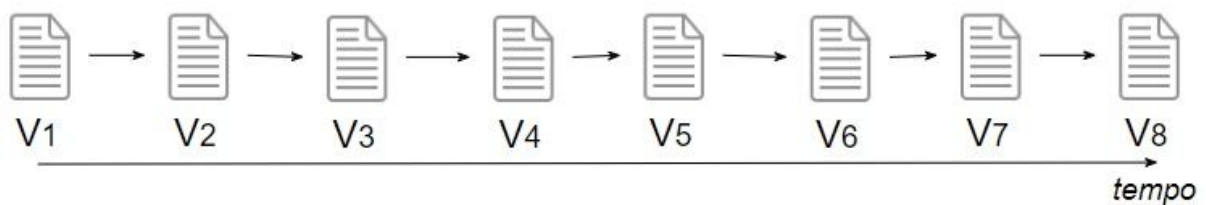


Fig. 4 - Struttura lineare delle versioni

Oppure un'unione delle due modalità.

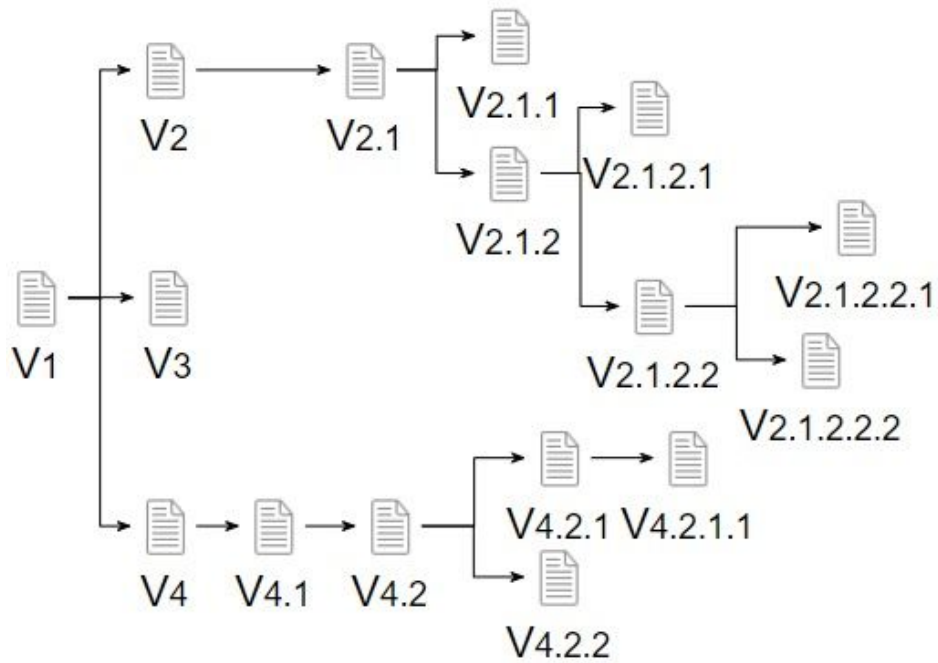


Fig. 5 - Struttura ad albero delle versioni

Per esprimere le differenze tra due versioni dello stesso documento si utilizzano i cambiamenti tra le due, queste differenze si esprimono con un insieme di modifiche. In entrambe le features si basano sulla condivisione delle modifiche di un documento.

Avere una buona struttura che esprima le modifiche sprecando poco spazio in memoria è fondamentale.

3. Lightly

3.1 Il progetto Lightly

Lightly è il progetto di un Change Tracking ed uno Stack UNDO-REDO che utilizza lo spazio minimo di memoria per salvare le modifiche sul documento e distinguere il tipo di modifica effettuata.

Queste due caratteristiche permettono di creare progetti futuri come il Versioning e Operational Transformation (vedi capitolo 2.2).

3.2 I tre livelli di modifica

Questo progetto si basa sul modello per le edit di documenti testuali ispirato dall'articolo di Angelo Di Iorio, Gianmarco Spinaci e Fabio Vitali chiamato Multi-layered edits for meaningful interpretation of textual differences [DSV19].

In questo articolo vengono presentati e definiti tre livelli per esprimere le modifiche sui documenti.

Il primo livello parte dall'esprimere cosa cambia prima e dopo la modifica come dei semplici inserimenti o eliminazioni di contenuto, questo livello è chiamato livello delle **modifiche meccaniche**.

Sopra questo livello si può interpretare la modifica distinguendola tra modifica al testo e modifica alla struttura e/o allo stile del documento, cioè i **nodi del documento** (vedi capitolo 4.2), questo livello di modifiche viene chiamato livello delle **modifiche strutturali**.

Infine sopra le modifiche strutturali viene definito il livello delle **modifiche semantiche**. Questo tipo di modifiche esprimono il significato semantico della modifica, rendendola più comprensibile per un essere umano.

Su lightly è stato scelto di utilizzare solo i primi due livelli di modifiche, il livello delle modifiche meccaniche e il livello delle modifiche strutturali.

Ho operato questa scelta in quanto la modifica semantica non comporta un utilizzo minore di memoria, diversamente dalle modifiche strutturali che rimuovono le parti non necessarie della modifica.

3.3 Modifica Meccanica

La modifica meccanica esprime i tipi di modifica più semplici senza distinguerli tra nodi e testo, considerando il tutto come testo (vedi capitolo 4.2).

Esistono due operazioni, INS per la aggiunta di contenuto e DEL per la cancellazione di contenuto.

La modifica meccanica ha al suo interno il contenuto della modifica, il tipo, la posizione della modifica, l'ID della modifica, il creatore e la data della modifica.

Modifica meccanica:		
ID	Identificativo della modifica	Integer
OP	Tipo di modifica	“INS”, “DEL”
POS	Posizione della modifica	Integer
CONTENT	Contenuto della modifica	String
BY	Autore della modifica	String
TIMESTAMP	Momento in cui è stata creata la modifica	String

Fig. 6 - Contenuto Modifica Meccanica

Le parti fondamentali sono l'OP, la POS e il CONTENT. Questi tre elementi hanno già tutte le informazioni necessarie per l'annullamento della modifica.

La modifica meccanica è la base per la modifica strutturale poiché la usa per esprimere il cambiamento effettuato sul documento.

Inoltre è la struttura meccanica sul quale il processo di UNDO e REDO opera perché solo questa ha le informazioni necessarie per annullare il cambiamento.

3.4 Modifica Strutturale

La modifica strutturale è ad un livello più alto delle modifiche meccaniche, aggrega una o più modifiche meccaniche per descrivere il cambiamento. In questo livello si introduce inoltre la distinzione tra il cambiamento su un nodo del documento o sul testo del documento.

Il nodi del documento sono quelli che definiscono lo stile del testo a livello grafico, come ad esempio il font, la grandezza del carattere, il peso, il colore etc. ne parleremo in modo più approfondito nel capitolo 4.2.

In alcuni casi avere una aggregazione di più modifiche meccaniche permette di risparmiare spazio. Ad esempio, da “*lorem ipsum dolor*” passo a “*lorem ipsum dolor*” creando un nuovo nodo attorno a “*ipsum*”, si può salvare la modifica come una INS di uno “” ed uno “” invece che salvarla come una DEL di “*ipsum*” e una INS di “ipsum”.

All'interno di una modifica strutturale sono presenti questi dati:

Modifica strutturale:		
ID	Identificativo della modifica	Integer
OP	Tipo di modifica	“TEXTINSERT”, “TEXTDELETE”, “TEXTREPLACE”, “INSERT”, “DELETE”, “WRAP”, “UNWRAP”, “REPLACE”, “CHANGE”
BY	Autore della modifica	String
TIMESTAMP	Momento in cui è stata creata la modifica	String
ITEMS	Array delle modifiche meccaniche associate	String
NEWMAP	Posizione del puntatore prima della modifica	Map
OLDMAP	Posizione del puntatore dopo la modifica	Map

Fig. 7 - Contenuto Modifica Strutturale

Ci sono 6 tipi di modifiche: TEXTINSERT, TEXTDELETE, TEXTREPLACE, INSERT, DELETE, WRAP, UNWRAP, REPLACE, CHANGE.

Le prime tre sono le modifiche effettuabili direttamente sul testo

- TEXTINSERT è la modifica di aggiunta di testo, al suo interno contiene una INS
- TEXTDELETE è la modifica di eliminazione di testo, al suo interno contiene una DEL
- TEXTREPLACE è la modifica di sostituzione di testo con altro testo, al suo interno contiene una DEL ed una INS

Le successive cinque sono le modifiche che riguardano i nodi

- INSERT è la modifica di creazione un nuovo nodo, al suo interno contiene una INS
- DELETE è la modifica di eliminazione di un nodo, al suo interno contiene una DEL
- WRAP è la modifica di creazione di un nodo attorno ad elementi già esistenti, al suo interno contiene 2 INS
- UNWRAP è la modifica di eliminazione del nodo lasciando però intatto il suo contenuto, al suo interno contiene 2 DEL
- REPLACE è la modifica di cambiamento del tipo di nodo, contiene 2 DEL e 2 INS

Infine c'è la CHANGE, questa è per le modifiche che riguardano sia nodi che testo ed è l'unica che non distingue i due tipi, al suo interno contiene una DEL ed una INS.

3.5 Annullamento della modifica

Una volta classificata la modifica e salvata dentro lo Stack UNDO-REDO è possibile annullarla con un UNDO, o dopo essere stata annullata rieseguita con un REDO.

Le due operazioni lavorano sulle modifiche meccaniche salvate all'interno della modifica strutturale.

Ci sono quattro casi possibili:

Tipo	UNDO	REDO
INS	Eliminazione del contenuto dallo stato	Aggiunta del contenuto nello stato
DEL	Aggiunta del contenuto nello stato	Eliminazione del contenuto dallo stato

Fig. 8 - Tabella dei casi di UNDO-REDO

Un UNDO su una INS ed una REDO su una DEL causano lo stesso cambiamento nello stato, cioè l'eliminazione del contenuto.

Invece un UNDO su una DEL ed una REDO su una INS causano lo stesso cambiamento nello stato, cioè l'aggiunta del contenuto.

3.6 Sessione di esempio

In questo capitolo verrà trattata una sessione di esempio per far vedere il comportamento del modello con un esempio reale.

1. Inizio



Fig. 9 - Lightly: Inizio

Questa è un'applicazione di esempio per il salvataggio delle modifiche lightly. A destra si può vedere tinyMCE. A sinistra si può vedere la rappresentazione grafica dello stato dello stack UNDO-REDO.

Al momento nel documento non si è scritto nulla, contiene solo un nodo che rappresenta la riga attuale.

2. Change

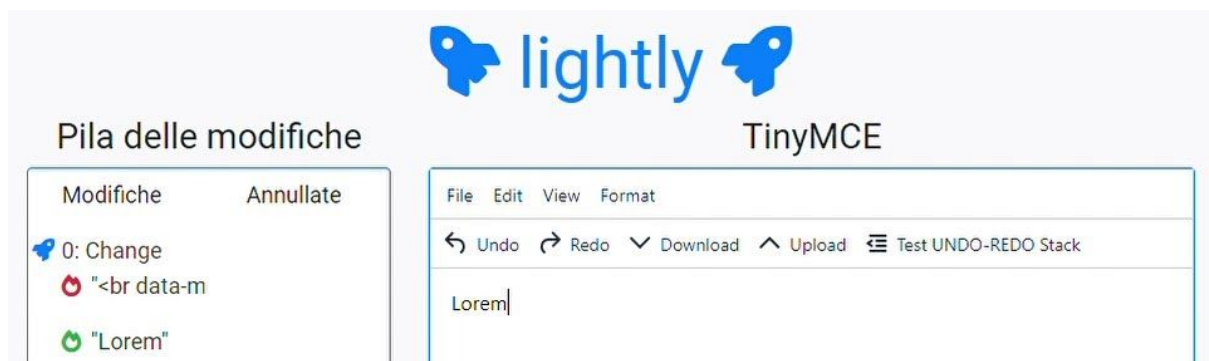


Fig. 10 - Lightly: Inserimento testo

L'utente scrive "Lorem" lo stato passa

da `<p><br data-mce-bogus="1"></p>`

a `<p>Lorem</p>`

Il contenuto del nodo `<p>` passa da tipo nodo ad tipo testo, quindi viene valutata come una modifica strutturale di tipo change.

3. Textinsert



Fig. 11 - Lightly: Inserimento testo

In seguito viene inserito " ipsum dolot", aggiungendo contenuto di tipo testo la modifica viene valutata come una modifica strutturale di tipo TEXTINSERT.

4. Correzione dell'errore

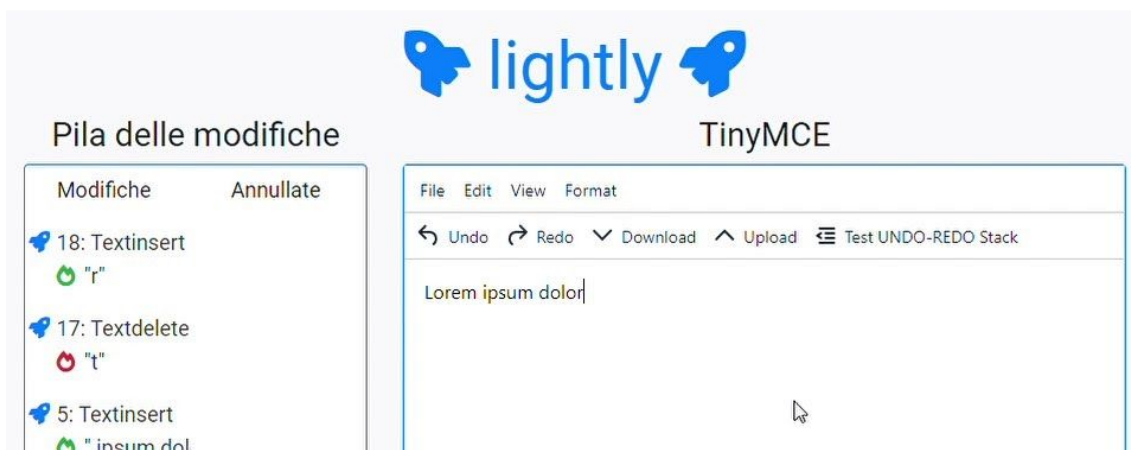


Fig. 12 - Lightly: Correzione errore

È stata corretta la “t” di “dolor” con una “r” trasformandola in “dolor”.

Vengono considerati come due cambiamenti uno di TEXTDELETE ed uno TEXTINSERT.

5. Fine frase

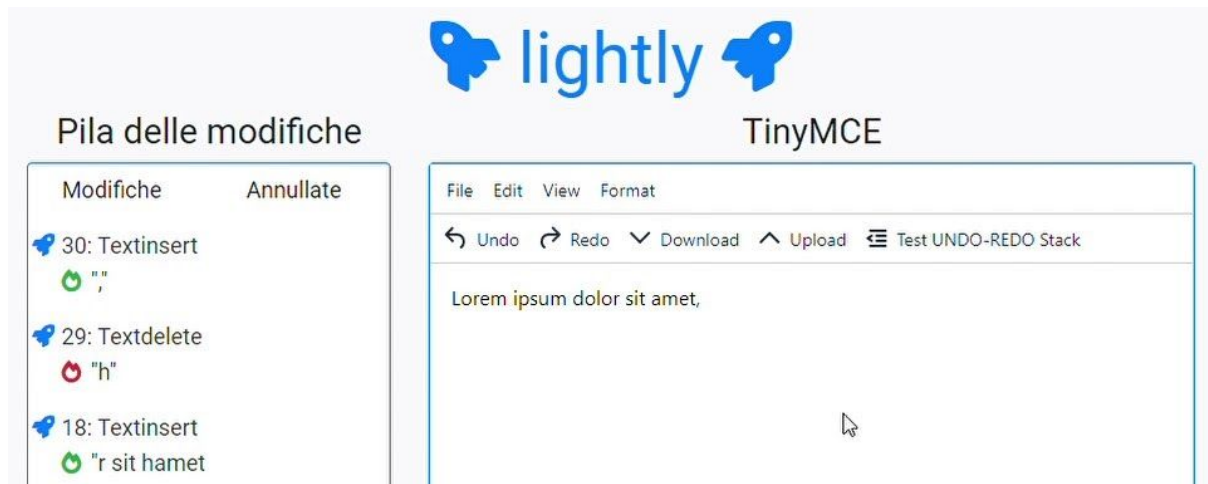


Fig. 13 - Lightly: Fine frase

In seguito è stato aggiunto “sit hamet” poi corretto con “sit amet” poi è stata aggiunta una virgola alla fine.

Il tutto viene tradotto come un aggiornamento della vecchia TEXTINSERT una nuova TEXTDELETE ed un'ultima TEXTINSERT.

6. Wrap

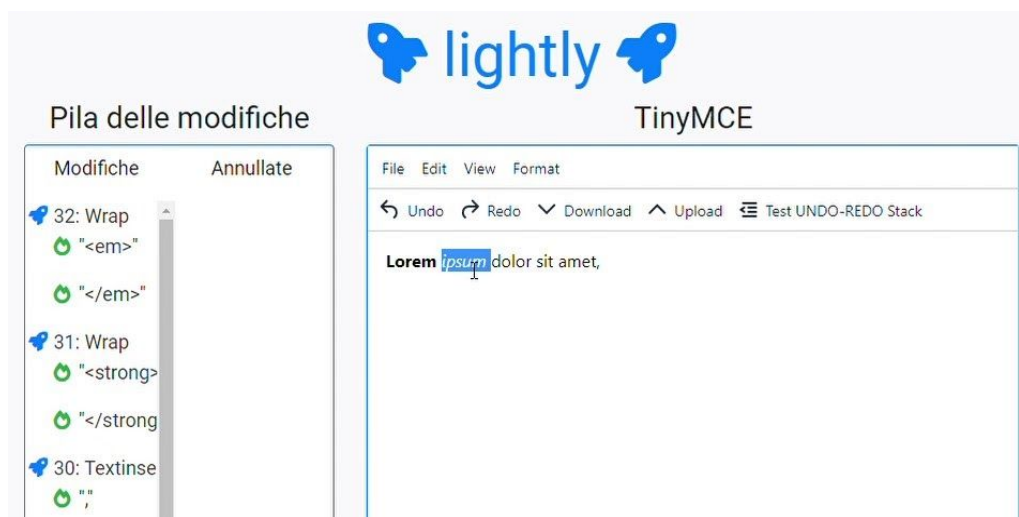


Fig. 14 - Lightly: Bold e Italic

In questo punto sono state eseguite due modifiche, nella prima è stato aggiunto il grassetto a lorem passando

da `<p>Lorem ipsum dolor sit amet,</p>`

a `<p>Lorem ipsum dolor sit amet,</p>`

È stato aggiunto il nodo `` all'interno del nodo `p` prendendo una parte del contenuto di `p` senza modificarlo.

Questo viene valutato come una modifica strutturale di tipo WRAP.

Nella seconda modifica anche a *“ipsum”* è stato aggiunto un nodo `` nella stessa modalità di prima ed è stata valutata come una modifica strutturale di tipo WRAP.

7. Unwrap

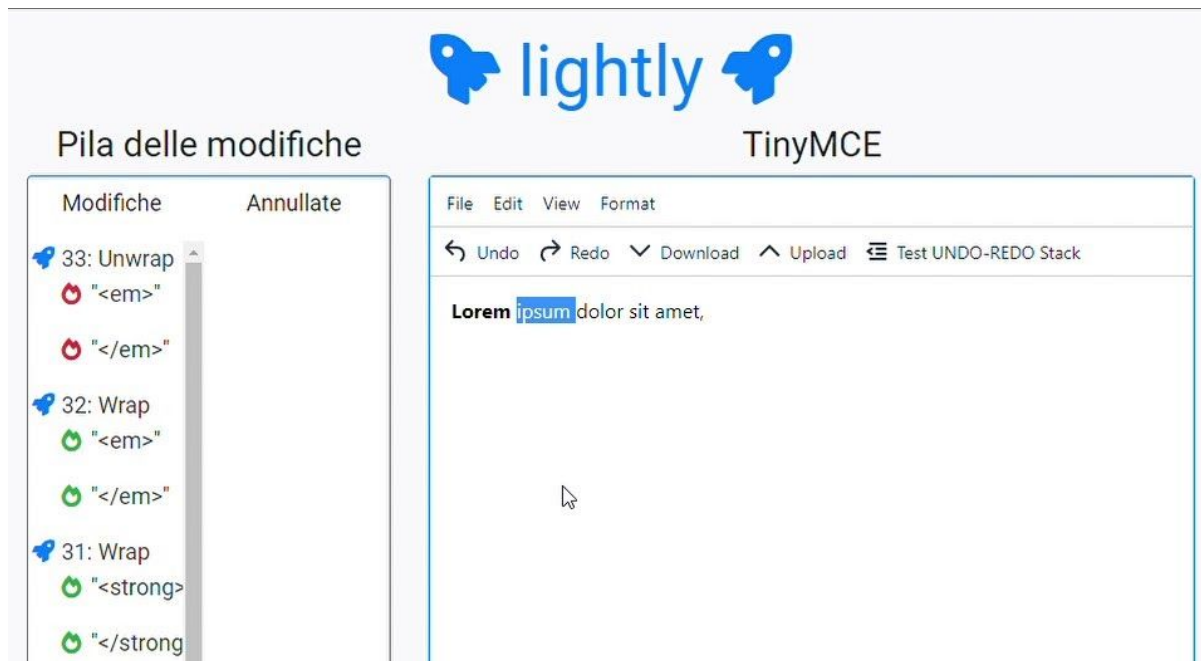


Fig. 15 - Lightly: Unwrap

In seguito il nodo `` è stato rimosso mantenendo però il suo contenuto, questo viene valutato come una modifica strutturale di tipo UNWRAP.

8. Undo di una modifica strutturale

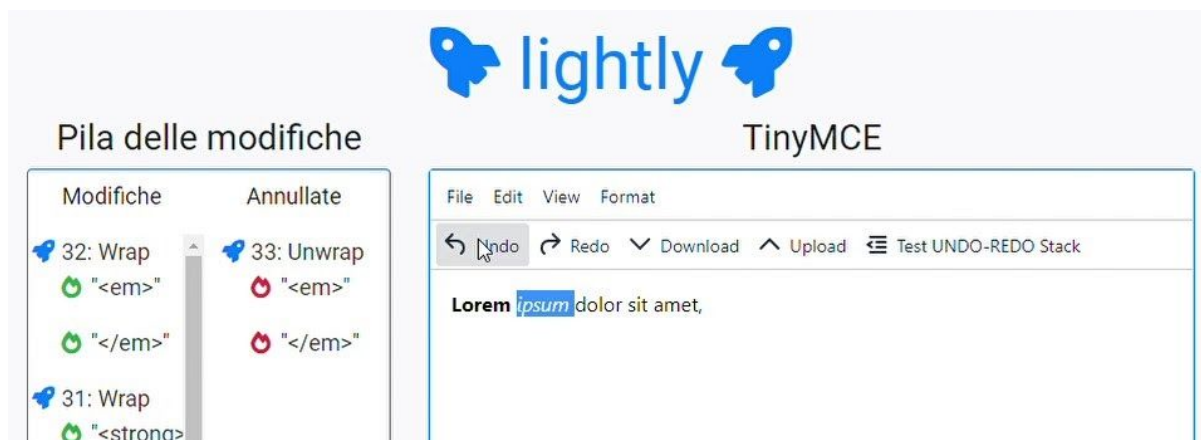


Fig. 16 - Lightly: Undo

L'utente ha premuto il pulsante Undo annullando l'ultima modifica rimettendo il nodo `` ad "ipsum".

Inoltre la modifica annullata viene spostata sulla seconda pila.

9. Undo di modifiche strutturali e testo

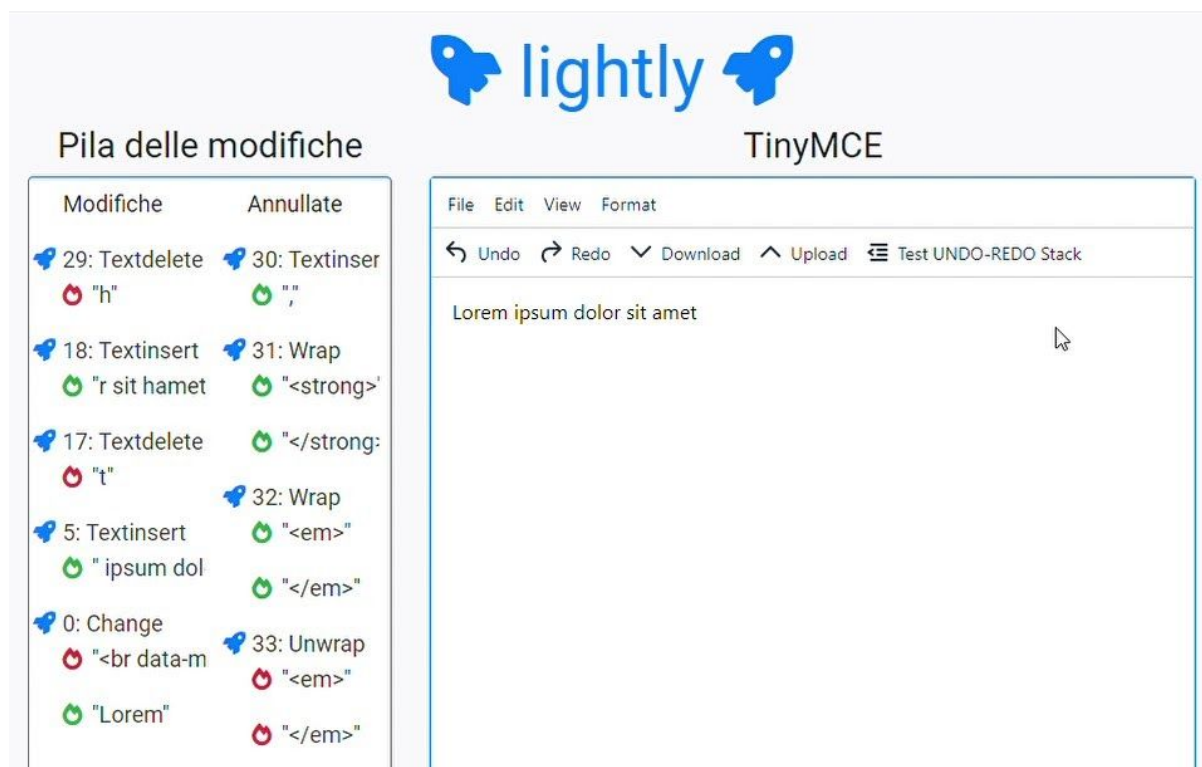


Fig. 17 - Lightly: Ulteriori Undo

Infine sono stati annullati il grassetto, l'italic e l'inserimento della virgola.

Da notare che sono state tutte spostate sulla seconda pila e che il documento è identico a quello prima di effettuare le modifiche annullate, questo è lo scopo del progetto.

4. Architettura di lightly

4.1 Introduzione

In questo capitolo viene spiegato a livello tecnico come è stata implementata l'ipotesi di un nuovo stack di UNDO e REDO per tinyMCE

Verranno affrontati i seguenti argomenti:

- Alberi
 - Nodi

Come TinyMCE gestisce nativamente il documento e che cos'è un nodo
 - Contenuto dell'albero

Come come si può leggere l'albero e quindi ottenere il suo stato.
- Struttura delle modifiche
 - Modifica meccanica e strutturale

Esempi più dettagliati sugli oggetti dove vengono salvate le modifiche meccaniche e strutturali.
 - Stack UNDO-REDO

Come vengono gestite le due pile e quali eventi scatenano il passaggio di elementi tra i due stack.
 - Esecuzione di UNDO e REDO

Come le modifiche vengono annullate o rieseguite.

- Range

- Posizione del cursore

Come leggere il cursore può generare discrepanze e come si è evitato per la selezione del range.

- Intervallo del range

Quali sono le regole che determinano la porzione del documento da controllare.

- Modifica

- Rilevazione del cambiamento

Cosa fa scatenare la ricerca di una modifica

- Cattura della modifica

L'algoritmo per la cattura della modifica e cosa estrapola

- Valutazione della modifica

Come si valuta il contenuto estrapolato e come lo si interpreta per capire il tipo di modifica

- Salvataggio della modifica

Conclusioni di cosa viene salvato all'interno della modifica strutturale

- Update della modifica

Quando bisogna aggiornare la modifica ed in cosa consiste

4.2 Albero

Su TinyMCE il documento ha una struttura che segue le regole del linguaggio HTML.

Il documento può essere letto come un albero che ha come un nodo principale il body del documento e tanti nodi figlio di vari tipi a seconda del tipo di elemento.

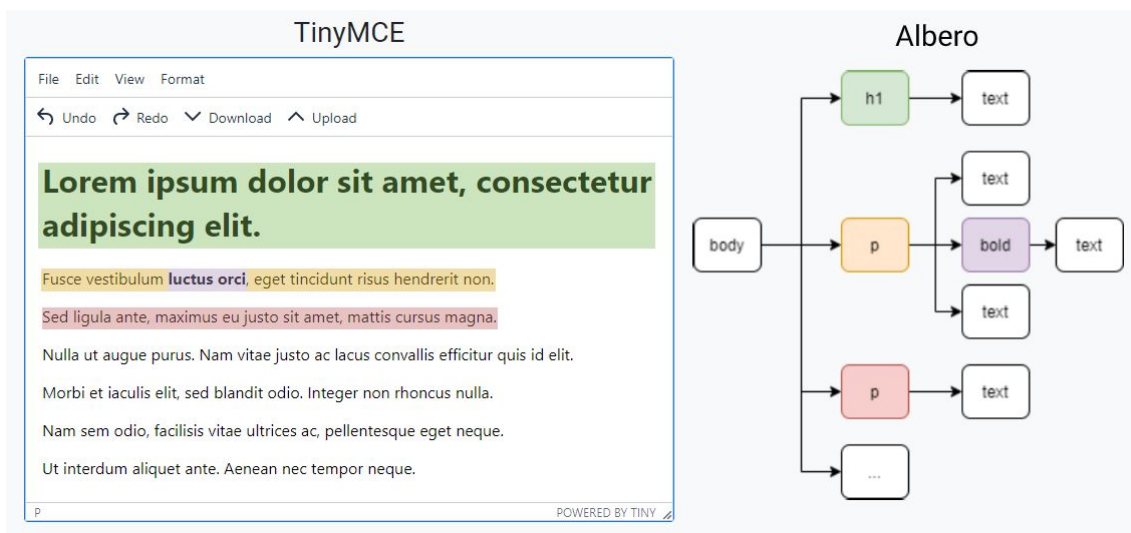


Fig. 18 - Struttura ad albero di TinyMCE

4.2.1 Nodi

Un nodo è quell'elemento che descrive lo stile del contenuto al suo interno e permette di avere una gerarchia ad albero all'interno del documento stesso.

Ogni nodo inizia e finisce in dentro il suo nodo padre, senza intersecarsi con i suoi fratelli.

4.2.2 Contenuto dell'albero

Visto che il documento è scritto in HTML si può interpretare lo stato come una stringa contenente i nodi ed il testo.

Questo metodo di lettura semplifica l'identificazione delle modifiche, perchè diventa un confronto tra due stringhe, quella prima della modifica, detta **oldState** e quella successiva alla modifica detta **newState**.

Anche la definire la posizione della modifica viene semplificata, perché viene identificata come la posizione del carattere all'interno della stringa.

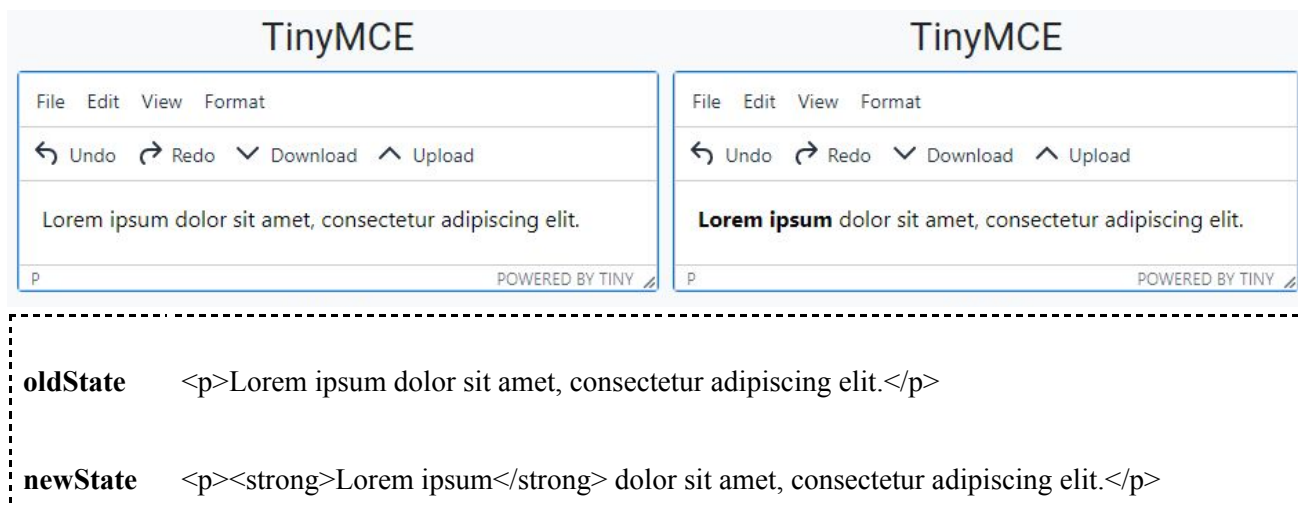


Fig. 19 - Stato di TinyMCE

Come si vede nell'immagine sopra al documento di destra è stato aggiunto il grassetto a Lorem ipsum, aggiungendo un elemento con tag strong con all'interno la stringa, questo si traduce come 2 inserimenti: uno a posizione 3 di "" ed un altro a posizione 22 "".

4.3 Struttura delle modifiche

La struttura delle modifiche è la base del progetto ed è basata sull'articolo [DSV19].

4.3.1 Modifica meccanica

La modifica meccanica è la modifica più a basso livello possibile, non distingue se il tipo di modifica è un nodo o del semplice testo.

Questo è un esempio della modifica meccanica:

```
MECH: {
  id: 16,
  op: "INS",
  pos: 71,
  content: "ipsum",
  by: "Francesco Fornari",
  timestamp: "2020-09-22T09:53:03.936Z"
}
```

Fig. 20 - Modifica Meccanica

OP può essere di due tipi:

INS: è il tipo di modifica per l’inserimento di nuovi elementi come testo o nodi.

DEL: è il tipo di modifica per la cancellazione di elementi come testo o nodi.

La struttura di esempio sopra descrive l’aggiunta della stringa “*ipsum*” a posizione 71.

4.3.2 Modifica strutturale

```
STRUCT: {
  id: 16,
  op: "WRAP",
  by: "Francesco Fornari",
  items: [
    0: {id: 19, op: "INS", pos: 3, content: "<strong>", by: "Francesco Fornari", ...}
    1: {id: 20, op: "INS", pos: 22, content: "</strong>", by: "Francesco Fornari", ...}
  ],
  newMap: {start: {...}, end: {...}},
  oldMap: {start: {...}, end: {...}},
  timestamp: "2020-09-23T21:57:54.682Z"
}
```

Fig. 21 - Modifica Strutturale

newMap e **oldMap** sono degli oggetti che esprimono la posizione del puntatore nel testo.

Ci sono nove tipi differenti di modifiche strutturali:

Codice	Dettaglio	Ordine delle modifiche meccaniche associate:
TEXTDELETE	Eliminazione di una stringa	► DEL
TEXTINSERT	Inserimento di una stringa	► INS
TEXTREPLACE	Cambiamento di una stringa	► DEL ► INS
DELETE	Eliminazione di un nodo	► DEL
INSERT	Inserimento di un nodo	► INS
WRAP	Annidamento di più nodi dentro un nodo, senza cambiarne il contenuto	► INS ► INS
UNWRAP	Eliminazione di un nodo promuovendo il suo contenuto ad un livello maggiore	► DEL ► DEL

REPLACE	Sostituzione di un nodo con un altro senza modificarne il contenuto	▶ DEL ▶ DEL ▶ INS ▶ INS
CHANGE	Sostituzione generica di una parte di testo con un'altra	▶ DEL ▶ INS

Fig. 22 - Modifiche Meccaniche all'interno delle Modifiche Strutturali

4.3.3 Stack UNDO-REDO

Lo stack UNDO-REDO sono due pile LIFO di modifiche strutturali.

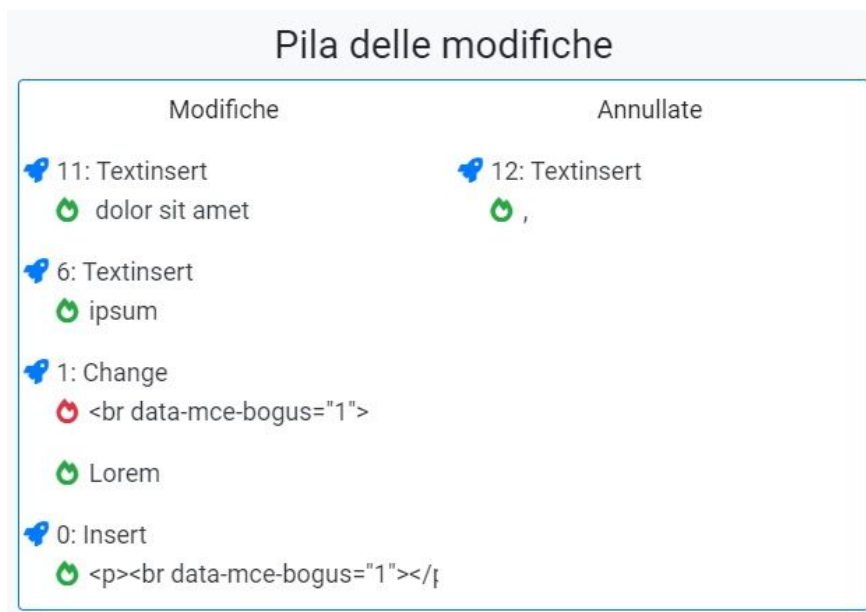


Fig. 23 - Pila delle modifiche

Quando viene eseguita una modifica la si cattura e la si salva in testa alla pila delle modifiche.

Quando viene eseguito un UNDO, ovvero l'annullamento della modifica, l'ultima modifica inserita viene annullata e spostata sulla pila delle annullate.

Quando viene eseguito un REDO, ovvero la riesecuzione della modifica, viene rieseguita e rispostata sulla pila delle modifiche.

Se invece viene eseguita una nuova modifica sul documento la pila delle modifiche annullate viene eliminata.

4.3.4 Esecuzione di UNDO e REDO

Come detto prima ogni modifica strutturale contiene un array di una o più modifiche nel caso di un UNDO vengono annullate dall'ultima fino alla prima invece in caso di REDO vengono rieseguite dalla prima all'ultima.

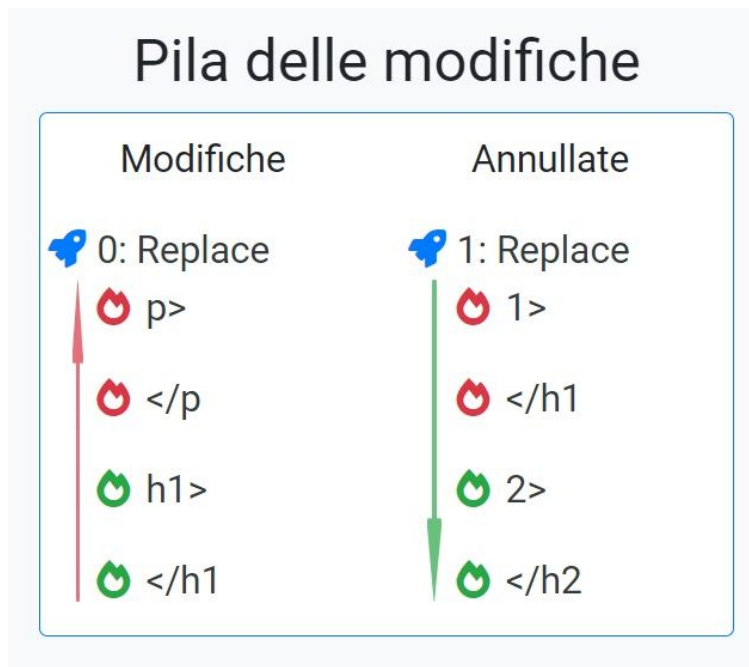


Fig. 24 - Verso di Undo e Redo

```
MECH: {  
  id: 2,  
  op: "INS",  
  pos: 2,  
  content: "h1>",  
  by: "Francesco Fornari",  
  timestamp:  
    "2020-09-22T10:52:01.926Z"  
}  
  
MECH: {  
  id: 0,  
  op: "DEL",  
  pos: 2,  
  content: "p>",  
  by: "Francesco Fornari",  
  timestamp:  
    "2020-09-22T10:52:01.926Z"  
}
```

Fig. 25 - I due tipi di Modica Meccanica

La struttura di sinistra descrive l'aggiunta della stringa "h1>" per annullare la modifica bisogna prendere lo stato attuale del documento, rimuovere la porzione che va da 2 a 5 e aggiornare lo stato con la nuova stringa. 5 equivale a pos sommato alla lunghezza di "h1>", cioè il content.

La struttura di destra descrive l'eliminazione della stringa "p>" a posizione 2, per annullarla bisogna tagliare lo stato attuale in due parti in posizione 2, aggiungere la stringa "p>" ed aggiornare lo stato.

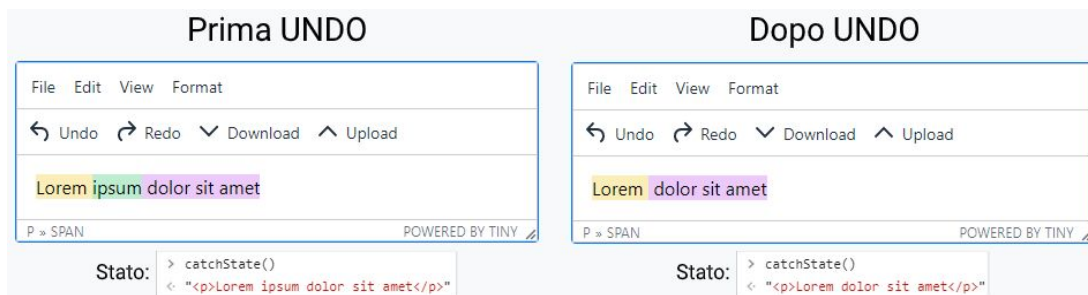


Fig. 26 - Prima e dopo UNDO

Questa è una rappresentazione grafica di cosa succede annullando l'inserimento della parola "ipsum".

4.4 Range

4.4.1 Posizione del cursore

Un nodo al suo interno può contenere sottonodi costituendo un sottoalbero, questo può rendere difficile il controllo del nodo per la cattura della modifica. Per questo motivo viene eseguito il controllo soltanto sui nodi figli del nodo radice, considerando i sottonodi come semplice testo.

Questa decisione fa perdere rapidità nella ricerca della modifica, visto che controlla delle parti che potrebbero essere escluse dalla ricerca, però fa acquistare alla ricerca consistenza evitando di dover costruire regole ad hoc per ogni evento.

Ad esempio nel caso del heading (titolo, sottotitolo) nonostante il puntatore stia in una qualsiasi posizione del sottoalbero tutto il nodo figlio della radice viene modificato.

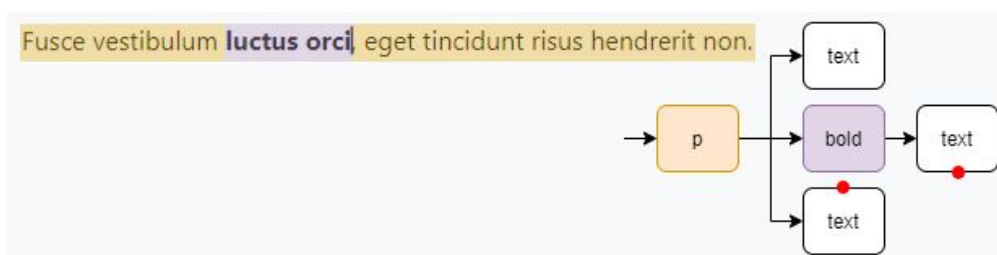


Fig. 27 - Posizione del puntatore

Inoltre valutare tutto il nodo principale evita il problema della posizione ambigua del cursore, ad esempio nell'immagine sopra se ci fosse un cursore posizionato alla fine del nodo in grassetto **bold**

anche se graficamente il risultato è identico potrebbe trovarsi o dopo l'ultimo carattere del figlio di bold oppure prima del primo carattere del fratello destro di bold, più precisamente nella figura sopra sono i pallini rossi.

Poi ci sono delle modifiche come la creazione del titolo, che fa cambiare a tutto il nodo principale il suo tag, se si valutasse solo il nodo dove risiede il cursore non si valuterebbe la parte corretta.

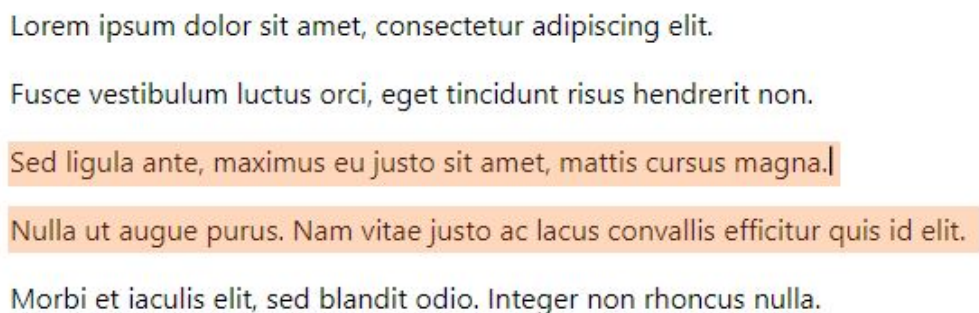
Selezionando tutto il sotto albero non si hanno questi tipi di ambiguità.

4.4.2 Intervallo del range

Per avere un controllo rapido ed efficiente si può utilizzare la posizione del cursore per sapere dove è stata inserita la modifica. In quanto il cursore è posizionato dove l'utente interagisce con il documento ed indica dove avvengono i cambiamenti.

Avere solo una piccola porzione da controllare aumenta in maniera considerevole le performance, in quanto evita di controllare tutto il documento ma solo solo una piccola parte.

La porzione del documento interessata verrà definita **Range**.



>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Fusce vestibulum luctus orci, eget tincidunt risus hendrerit non.
Sed ligula ante, maximus eu justo sit amet, mattis cursus magna.
Nulla ut augue purus. Nam vitae justo ac lacus convallis efficitur quis id elit.
Morbi et iaculis elit, sed blandit odio. Integer non rhoncus nulla.

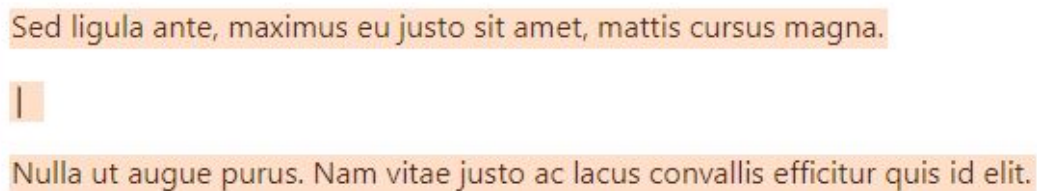
Fig. 28 - Range sul documento

Il range va dal nodo dove era presente il cursore prima della modifica al nodo successivo alla posizione del cursore dopo la modifica.

Nell'esempio mostrato sopra è stato aggiunto un punto alla fine della frase, il nodo dove era ospitato il cursore prima e dopo la modifica corrisponde, quindi vengono controllati solo due nodi evitando di controllare gli altri tre.

La ricerca viene eseguita in questo range, perchè qualsiasi tipo di modifica può riguardare al massimo questo range di nodi.

Visto che prima della cattura e dell'individuazione del codice è difficile determinare il tipo di modifica effettuato, ho deciso di controllare sempre questo range per avere più solidità nella ricerca.



```
Sed ligula ante, maximus eu justo sit amet, mattis cursus magna.  
|  
Nulla ut augue purus. Nam vitae justo ac lacus convallis efficitur quis id elit.
```

Fig. 29 - Range dopo un invio

4.5 Modifica

Intercettare la modifica, valutarla e salvarla nello stack è il cuore del progetto. Avere una buona individuazione della modifica rende il progetto rapido, invece avere una buona valutazione rende lo Stack UNDO-REDO leggero e parlante.

4.5.1 Rilevazione del cambiamento

Avere una buona rilevazione del cambiamento è fondamentale per avere una cattura della modifica sensata e priva di errori.

Lightly è in ascolto su 4 eventi:

1. Prima che un evento avvenga
2. Appena un pulsante sulla tastiera è stato premuto
3. Dopo l'avvenimento dell'evento
4. Quando si rilascia il pulsante della tastiera

I primi due eventi avvengono prima del cambiamento e servono a lightly per salvarsi la posizione del cursore prima della modifica.

Gli ultimi due eventi servono a `lightly` per lanciare il rilevamento del cambiamento, passandogli come parametri la posizione del cursore prima del cambiamento per salvarlo all'interno della modifica strutturale e per determinare il range da controllare.

4.5.2 Cattura della modifica

Per catturare la modifica si confrontano due stati: quello precedente alla modifica e quello successivo alla modifica.

I due stati sono letti come due stringhe e verrà controllata solo la zona data dal range.

oldState: Lorem dolor sit amet, consectetur adipiscing elit.

newState: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

oldState: Lorem dolor sit amet, consectetur adipiscing elit.

newState: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

oldState: Lorem dolor sit amet, consectetur adipiscing elit.

newState: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

ADD: "ipsum "

DEL: ""

Fig. 30 - Controllo della stringa

Nell'esempio qui sopra è stato aggiunto la stringa "ipsum" alla frase.

Il sistema rileva un cambiamento e si susseguono una serie di eventi:

1. Si carica la posizione del puntatore precedente al caricamento
2. Si salva il nuovo stato chiamato `newState`
3. Si determina il range con le modalità definite nei paragrafi precedenti
4. Si controlla la porzione di stringa col cambiamento

5. Si estrae la parte rimossa e quella aggiunta
6. Si valutano le due parti per riconoscere il tipo di modifica effettuato
7. Si salva la modifica sullo stack di UNDO

Il controllo della modifica avviene carattere per carattere e viene eseguito in due versi:

- Il primo è da sinistra verso destra, in figura è colorato in azzurro. Finché i caratteri combaciano continua da avanzare e si ferma quando viene individuato un carattere diverso
- Il secondo controllo va da destra verso sinistra, in figura è colorato di giallo. Continua fino a che non trova anche questo un carattere differente o arriva al carattere del primo controllo.

A questo punto ci sono due stringhe estratte, quella di oldState e quella di newState.

La prima viene valutata come la stringa eliminata dallo stato, la seconda invece come la stringa aggiunta allo stato.

C'è da notare che una delle due stringhe potrebbe essere vuota, ci sono 3 casi:

- La stringa eliminata è vuota e la stringa aggiunta contiene elementi. In questo caso l'utente sta aggiungendo parti al documento.
- La stringa eliminata contiene elementi e la stringa aggiunta è vuota. In questo caso l'utente sta eliminando parti del documento.
- Entrambe le stringhe contengono elementi. Questo è il caso più vario, l'utente in questo momento potrebbe modificare dei nodi già esistenti o ne sta creando di nuovi spostando del testo al loro interno senza modificarlo oppure potrebbe sostituire parti di testo con altro testo.

4.5.3 Valutazione della modifica

Una volta ottenute le 2 stringhe si analizzano con una regex per capire se all'interno della stringa sono presenti dei nodi o del semplice testo, in seguito vengono smistati nei 9 tipi definiti prima:

TEXTINSERT	Se non sono presenti nodi e solo la stringa di add è popolata
TEXTDELETE	Se non sono presenti nodi e solo la stringa di del è popolata
TEXTREPLACE	Se non sono presenti nodi e tutte e 2 le stringhe sono popolate
INSERT	Se sono presenti nodi e solo la stringa di add è popolata
DELETE	Se sono presenti nodi e solo la stringa di del è popolata
WRAP	Se sono presenti dei nodi della stringa di add ed il suo contenuto corrisponde a del
UNWRAP	Se sono presenti dei nodi della stringa di del ed il suo contenuto corrisponde ad add
REPLACE	Se sono presenti nodi in tutte e due le stringhe ed il contenuto dei 2 nodi combaciano
CHANGE	In tutti gli altri casi

Fig. 31 - Regole per la determinazione ei tipi strutturali

CHANGE è il nodo ombrello che copre tutti gli altri casi, ed è quello che fa da ponte tra le modifiche sul testo e quelle sui nodi

4.5.4 Salvataggio della modifica

Un volta determinato il tipo di modifica vengono create le modifiche meccaniche.

Queste vengono salvate nell'array items della modifica strutturale. Poi vengono salvate le posizioni del cursore prima e dopo la modifica in oldMap e newMap. Viene salvata il tipo di modifica in OP e tutte le altre info come la data della modifica, il creatore e l'identificativo della modifica.

4.5.5 Update delle modifiche

In alcuni casi si possono unire più modifiche dello stesso tipo per formare modifiche più articolate, ad esempio se scrivo una frase con più caratteri o eseguo una cancellazione di più parti di testo se rispettano certi criteri ha senso che più modifiche vengano unite tra loro.

I criteri per unire più modifiche sono:

- I tipi delle due modifiche sono compatibili tra di loro

Modifica già presente	Modifica aggiunta
TEXTINSERT	TEXTINSERT
TEXTDELETE	TEXTDELETE
TEXTREPLACE	TEXTINSERT
CHANGE	TEXTINSERT

Fig. 32 - Compatibilità degli update delle modifiche

- L'intervallo di tempo tra le due modifiche è sotto una soglia di 3 secondi
- La posizione delle due modifiche è compatibile
 - TEXTINSERT: La posizione della modifica in entrata corrisponde alla posizione dell'ultimo carattere nella modifica già presente
 - TEXTDELETE: La posizione della modifica in entrata corrisponde alla posizione precedente al primo carattere nella modifica già presente

5. Valutazione

Come metro di valutazione per le prestazioni del progetto ho esaminato il consumo di RAM da parte del browser.

Come ambiente di prova è stato deciso di utilizzare un Mac e come Browser Web Safari. Sono stati eseguiti tre esperimenti differenti, con e senza lightly, per studiare l'impatto sulla ram del programma.

5.1 Primo test

Il primo test è stato eseguito su un testo “fantoccio” di 10 MegaByte. Il testo è molto semplice con piccoli nodi di tipo <p> alternati ad un invio vuoto. Per 40 volte è stata eseguita una trasformazione del nodo in tipo <h1>.

Questi sono i dati e i grafici di utilizzo con e senza lightly del primo test.

Consumo ram con lightly					
Avvenimenti	Totale	Reale	Virtuale	Condivisa	Privata
Base	6310,7	112,6	6031,4	112,9	53,8
Inserimento	6310,7	112,6	6031,4	112,9	53,8
Fine delle modifiche	6362,3	124,9	6072,3	109,1	56,0

Fig. 33 - Primo test con lightly

Consumo ram senza lightly

Avvenimenti	Totale	Reale	Virtuale	Condivisa	Privata
Base	6390,8	142,0	6082,56	111,8	54,4
Inserimento	6389,8	141,5	6082,56	109,9	55,8
Fine delle modifiche	6394,4	142,1	6082,56	110,1	59,6

Fig. 34 - Primo test senza lightly

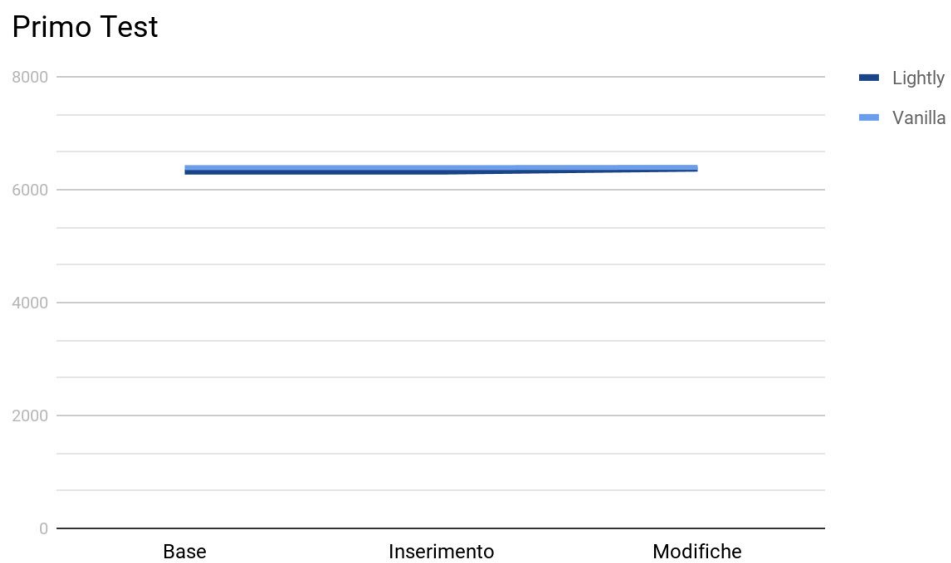


Fig. 35 - Grafico primo test

5.2 Secondo test

Il secondo test è stato eseguito su un documento reale sempre di 10 MegaByte.

Le modifiche sono state create a mano sul documento cercando di eseguire le stesse azioni in contemporanea.

Questi sono i dati e i grafici di utilizzo con e senza lightly del secondo test.

Consumo ram con lightly					
Avvenimenti	Totale	Reale	Virtuale	Condivisa	Privata
Base	6305,4	102,4	6051,8	102,1	49,1
Inserimento	6500,9	170,0	6133,8	101,1	96,0
Parte in Bold	6447,7	170,7	6123,5	101,6	51,9
Parte in Italic	6447,3	170,6	6123,5	101,7	51,5
Parte in Bold	6447,6	170,8	6123,5	101,7	51,6
Parte tagliata	6426	149,9	6123,5	101,7	50,9
Parte incollata	6405,6	149,9	6103	101,7	51,0

Fig. 36 - Secondo test con lightly

Consumo ram senza lightly					
Avvenimenti	Totale	Reale	Virtuale	Condivisa	Privata
Base	6317,6	105,7	6072,3	81,7	57,9
Inserimento	6614,6	207,2	6195,2	83,2	129,0
Parte in Bold	6556,6	207,3	6195,2	83,5	70,6
Parte in Italic	6556,2	207,1	6195,2	83,7	70,2
Parte in Bold	6556,5	207,2	6195,2	83,7	70,4
Parte tagliata	6544,3	208,0	6195,2	83,7	57,4
Parte incollata	6544,2	208,1	6195,2	83,7	57,4

Fig. 37 - Secondo test senza lightly

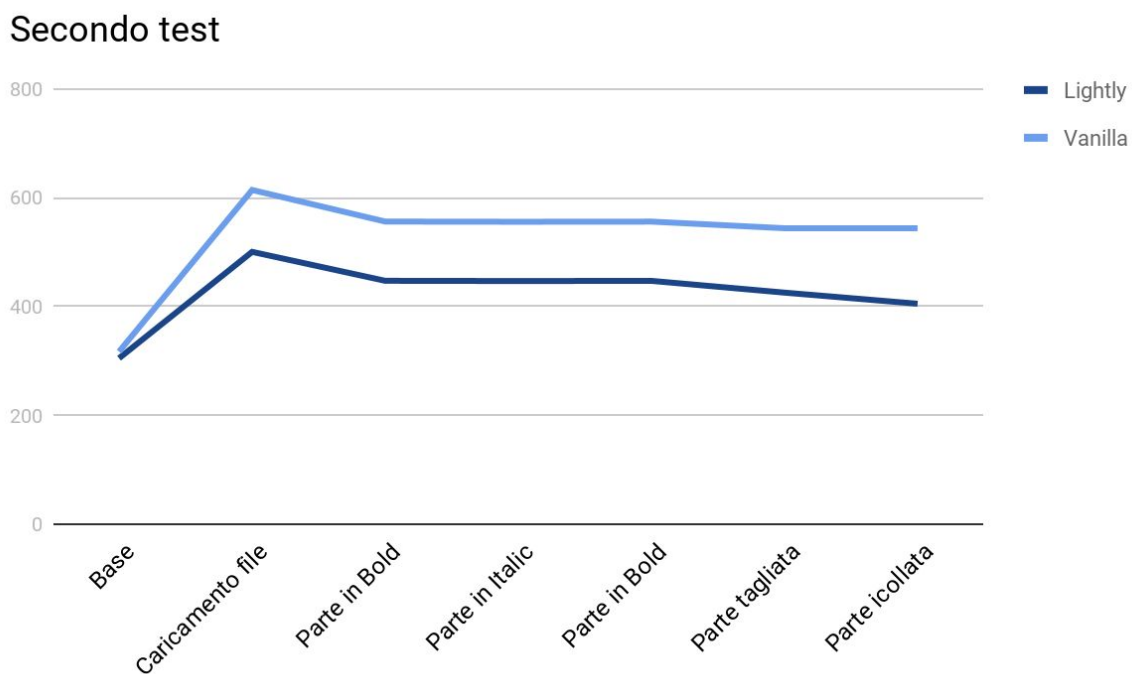


Fig. 38 - Grafico secondo test

5.3 Terzo test

Questo test è stato eseguito sul file precedente, è stato modificato copiando ed incollando a mano più volte una parte significativa del documento, in seguito è stata eseguita una sequenza di UNDO per riportare il documento allo stato originale.

Questi sono i dati e i grafici di utilizzo con e senza lightly del terzo test.

	Con Lightly		Senza Lightly	
Prima del test	1733.2 MB	107.1 MB 0.39 GB, 27.5 MB, 55.7 MB 108.4 MB, 0.91 GB, 82.1 MB, 52.4 MB	1723.8 MB	92.9 MB, 0.37 GB, 27.0 MB, 47.5 MB 116.8 MB, 0.93 GB, 82.9 MB, 56.7 MB
Dopo inserimento del file	3226.7 MB	488.4 MB, 0.82 GB, 28.3 MB, 441.6 MB 210.7 MB, 1.03 GB, 82.1 MB, 125.6 MB	2816 MB	395.6 MB, 0.71 GB, 27.0 MB, 308.1 MB 205.3 MB, 1.03 GB, 83.3 MB, 56.7 MB
Alla fine del test	6505.9 MB	1.63 GB, 1.99 GB, 27.8 MB, 1.48 GB 211.2 MB, 1.03 GB, 82.3 MB, 54.6 MB	4648.8 MB	1.00 GB, 1.31 GB, 26.9 MB 934.1 MB 207.1 MB, 1.03 GB, 82.2 MB, 58.5 MB
Dopo gli undo	7105.9 MB	1.96 GB, 2.28 GB, 27.5 MB, 1.46 GB 211.2 MB, 1.03 GB, 82.7 MB, 54.5 MB	TEST FALLITO	

Fig. 39 - Terzo test con e senza lightly

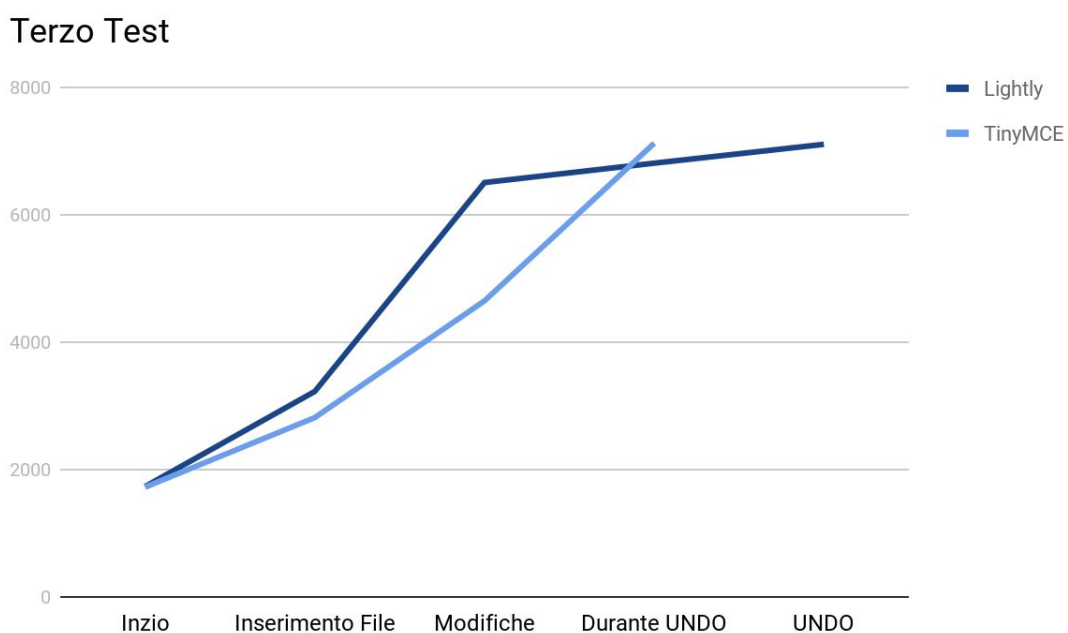


Fig. 40 - Grafico terzo test

5.4 Conclusioni del test

Come si può vedere la differenza di consumo di memoria tra le prove con lightly e le prove con la versione vanilla è minima.

La differenza sostanziale tra le due versioni la si può notare esaminando il terzo test. Come prima descritto se si prende in considerazione il consumo di memoria è simile, ma si è giunti al termine del test solo nella versione che utilizza lightly. Nella versione vanilla non è stato possibile concludere il test in quanto il sistema ha avuto un picco di consumo delle risorse del computer andando in stallo.

In seguito la prova è stata fatta anche su un pc windows usando Chrome come Web Browser, in questo caso il pc non è andato in stallo, però ha impiegato una quantità importante di tempo e risorse rispetto a lightly.

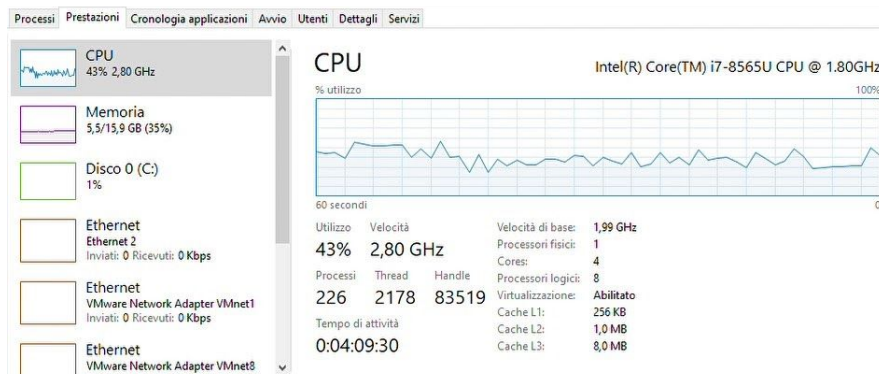


Fig. 41 - Grafico risorse utilizzate durante gli UNDO con lightly

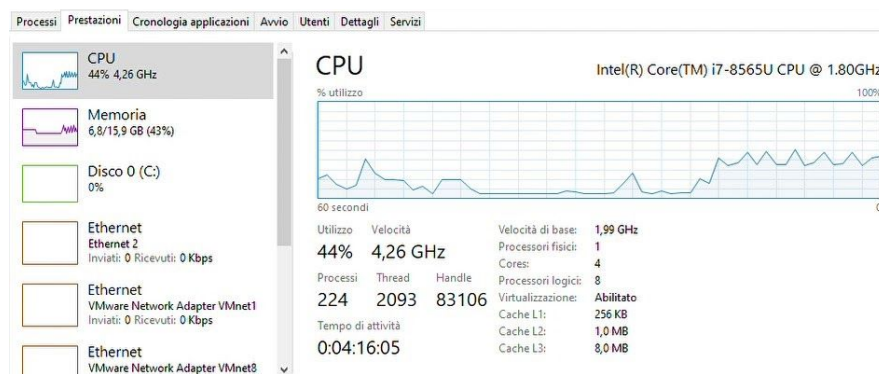


Fig. 42 - Grafico risorse utilizzate durante gli UNDO con TinyMCE

6. Conclusione

Inizialmente la tesi su cui si basa il progetto era che utilizzare un sistema di Change Tracking e uno Stack UNDO-REDO basato sull'articolo di Di Iorio, Spinaci e Vitali - Multi-layered edits for meaningful interpretation of textual differences, che sia più leggero sulla RAM e che permetta a TinyMCE di essere utilizzato anche per l'editing di documenti più pesanti in maniera fruibile.

Dopo l'analisi dei test si può affermare che il sistema, con e senza l'utilizzo di lightly, ha una differenza nell'impiego della memoria trascurabile.

In compenso si nota un netto aumento della richiesta delle risorse durante le operazioni di salvataggio delle modifiche e di avvenimenti UNDO-REDO effettuati su parti del documento con una struttura ad albero molto ricca di nodi. Addirittura il test eseguito sulla versione "vanilla" di TinyMCE manda in stallo la pagina web.

La mia teoria a proposito dell'utilizzo maggiore di risorse è che TinyMCE usa un sistema di memorizzazione delle modifiche diverso da lightly. La teoria è che non si basi sull'interpretazione dell'albero del documento come una stringa ma come i nodi effettivi del documento.

La mia opinione è che TinyMCE utilizza un metodo di interpretazione della modifica sull'albero che richiede un maggiore impiego di risorse, in quanto deve effettivamente esplorare i vari sottoalberi per salvare la modifica. Inoltre durante gli avvenimenti UNDO-REDO il metodo di salvataggio delle modifiche di TinyMCE è evidentemente molto più dispendioso rispetto a lightly.

Un mio suggerimento per progetti futuri, è che si dovrebbe indagare ulteriormente per trovare le cause di un consumo eccessivo di risorse al fine di creare nuovi progetti per ridurre il loro utilizzo.

La mia conclusione a proposito della tesi iniziale si è mostrata parzialmente corretta, poiché l'impiego di RAM non è risultato significativamente migliorato, ma l'utilizzo della struttura di modifiche a tre livelli rende possibile lavorare su TinyMCE con documenti pesanti.

Inoltre posso affermare che il modello a tre livelli, anche se introdotto parzialmente all'interno di lightly, funziona ed è effettivamente applicabile per descrivere le modifiche di un Word Processor.

Questa conclusione rende possibile l'utilizzo di lightly e del modello a tre livelli come base di partenza per futuri progetti al fine di introdurre l'Operational Transformation ed il Versioning su TinyMCE.

Inoltre, un sistema come lightly, che non è basato su una particolare caratteristica di TinyMCE, ma è principalmente basato sull'interpretazione dell'albero HTML, rende il progetto trasportabile su altri Web Word Processor con relativa semplicità.

Sarebbe sufficiente che il WWP abbia queste tre caratteristiche:

- Utilizza HTML per rappresentare il documento.
- Ha un metodo per accedere all'albero HTML.
- Ha un metodo per scrivere sull'albero HTML.

Quindi se in futuro si baseranno nuovi progetti di Versioning e di Operational Transformation su lightly sarebbe possibile pensare di esportare tutto il pacchetto su diversi Web Word Processor.

7. Bibliografia

- [DSV19] Angelo Di Iorio, Gianmarco Spinaci, and Fabio Vitali. 2019. “Multi-layered edits for meaningful interpretation of textual differences.” *Association for Computing Machinery*. 10.1145/3342558.3345406.
<https://dl.acm.org/doi/10.1145/3342558.3345406>
- Gadea, Cristian, Bogdan Ionescu, and Dan Ionescu. 2020. “A Control Loop-based Algorithm for Operational Transformation.” *IEEE 14th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. 10.1109/SACI49304.2020.9118822.
<https://ieeexplore.ieee.org/document/9118822>
- [TH06] Thomas Haigh. 2006. “Remembering the Office of the Future: The Origins of Word Processing and Office Automation.” *IEEE Annals of the History of Computing*. 10.1109/MAHC.2006.70.
<https://ieeexplore.ieee.org/document/4042483>
- Vitali, Fabio. 2019. *Un modello a livelli per le edit su documenti testuali*.
<https://docs.google.com/document/d/1383XYQRljRONcRLADtpSQb418bdyfZnZXka2xEiL70o/edit?usp=sharing>.
- Vitali, Fabio, Angelo Di Iorio, and Francesco Poggi. n.d. *Versioning al DASPLab*.
https://drive.google.com/file/d/18FufptI_BtSAMenz1Kr22QbzIUL8-9lZ/view?usp=sharing.