



Indice

- 1.** Team information and Project guidelines (pag. 3)
- 2.** List of Challenging/Risky requirements or tasks (pag. 5)
- 3.** Collection Requirements
 - **3.1:** Functional Requirements (pag. 7)
 - **3.2:** Scenari FR and UC description (pag. 10)
 - **3.3:** Non Functional Requirements (pag. 12)
- 4.** Content (pag. 13)
- 5.** Assumptions (pag. 14)
- 6.** Analysis Model (pag. 15)
- 7.** Component Diagram (pag. 17)
- 8.** Sequence Diagram (pag. 20)
- 9.** Class Diagram (pag. 24)
- 10.** Design Decision (pag. 27)
 - **10.1:** DBMS Design decision (pag. 28)
- 11.** How FR and NFR are satisfied by design (pag. 29)
- 12.** Effort Recording (pag. 30)
- 13.** GUI specification + code (pag. 31)
- 14.** ER Design (pag. 59)
- 15.** Code Engine (pag. 60)
- 16.** Code DBMS (pag. 69)
- 17.** About Us (pag. 72)

1. Team Information and Project guidelines

Date	08/02/2016
Deliverable	3
Team (Name)	SQLR

Team Members			
Name & Surname		Matriculation Number	E-mail address
Loris Fratarcangeli		227382	<i>loris94fr@gmail.com</i>
Patrizio Pezzilli		227636	<i>patriziopezzilli@gmail.com</i>
Stefano Cortellessa		227630	<i>stefano.cortellessa@student.univaq.it</i>
Valentino Giosaffatte	Di	232411	<i>valentino.digiosaffatte@student.univaq.it</i>
Riccardo A. Di Prinzi	o	229032	<i>riccardoarmando.diprinzio@student.univaq.it</i>
Flavio Furia		229034	<i>flavio.furia@student.univaq.it</i>
Luca Ferrari		229666	<i>luafe131@gmail.com</i>

Project Guidelines

This page provides the Guidelines to be followed when preparing the report for the Software Engineering course. You have to submit the following information:

- *This Report*
- *Diagrams (Analysis Model, Component Diagrams, Sequence Diagrams, Entity Relationships Diagrams)*
- *Effort Recording (Excel file)*

Important:

- *document risky/difficult/complex/highly discussed requirements*
- *document decisions taken by the team*
- *iterate: do not spend more than 1-2 full days for each iteration*
- *prioritize requirements, scenarios, users, etc. etc.*



2. List of Challenging/Risky Requirements or Tasks

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
<i>Separazione tra upload e creazione di un albero</i>	18/11/15	18/11/15	Il team ha deciso di assegnare alla GUI tre funzionalità: una per creare l'albero su file universale, una seconda per caricare quest'ultimo su database ed infine una terza per richiamare l'engine del calcolo.
<i>Decisione sul numero e formato del file</i>	18/11/15	26/11/15	Il team, grazie alle risposte del customer, ha deciso di utilizzare un singolo file in formato xml, da utilizzare durante tutte le fasi di esecuzione del sistema.
<i>Assegnazione valori numerici agli attributi</i>	18/11/15	26/11/15	Abbiamo optato per due opzioni a disposizione del customer: campi vuoti per inserimento di numeri a scelta, oppure un pulsante “random” che premuto, crea valori casuali.
<i>Assegnazione valori numerici agli attributi</i>	10/12/15	11/12/15	Il team ha rivisitato l’idea precedente e deciso, anche grazie alla discussione con il customer, di assegnare valori casuali ai vertici e agli archi.
<i>Introduzione file system in component e sequence diagram</i>	16/12/15	16/12/15	Il team per convenienza e semplicità, ha optato per introdurre la componente file system per una maggiore fluidità del sequence diagram.

<i>Struttura DMBS</i>	13/01/16	14/01/16	Il team ha optato per lo splittamento della tabella contentente l'OBJuid, in modo da gestire meglio il DMBS.
<i>Struttura dati Albero</i>	11/01/16	14/01/16	Il team ha optato per una struttura ad Array (vedere documentazione Engine)
<i>Collegamento DBMS-GUI & GUI-Engine</i>	01/02/16	5/02/16	Risolto aggiungendo gli attributi Split Size, Depth e Id (come chiave primaria) per risolvere errore nell'esecuzione del calcolo



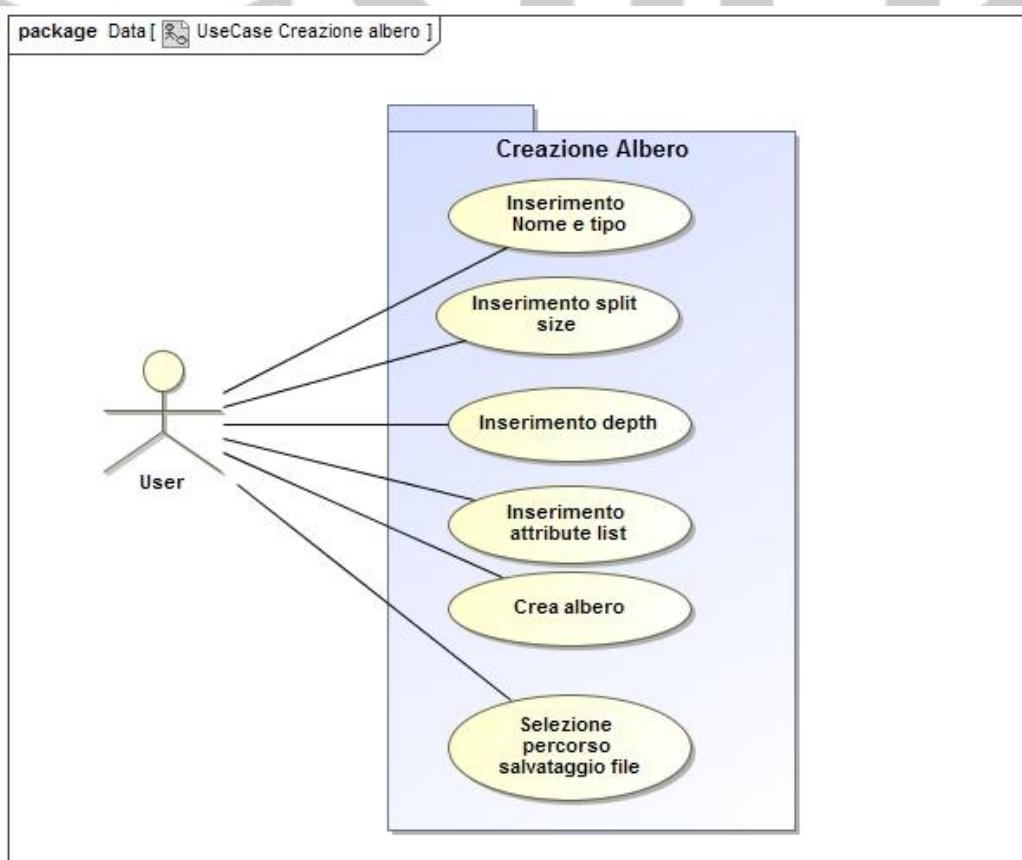
3. Requirements Collection

3.1 Functional Requirements

I requisiti funzionali che seguono verranno prioritizzati su scala da 1 a 5 , in base alla loro importanza(1= poco importante , 5= molto importante).

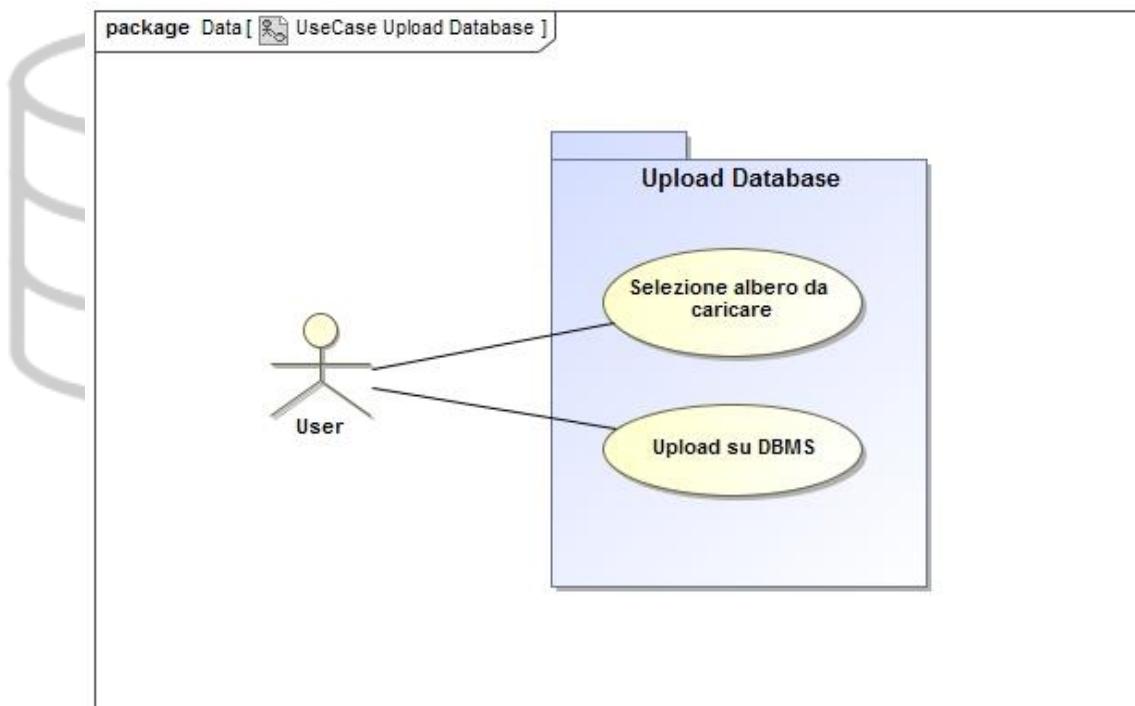
GUI:

1. Il sistema dovrà essere in grado di creare un albero a partire da dati immessi e successivamente dovrà essere in grado di calcolare il numero di nodi del percorso e la somma dei relativi attributi.
2. La GUI dovrà garantire all'utente di selezionare all'apertura la funzione interessata: Creazione di un nuovo albero, Upload di un file albero su un database, esecuzione del calcolo delle somme degli attributi di un albero.
3. Nello specifico:



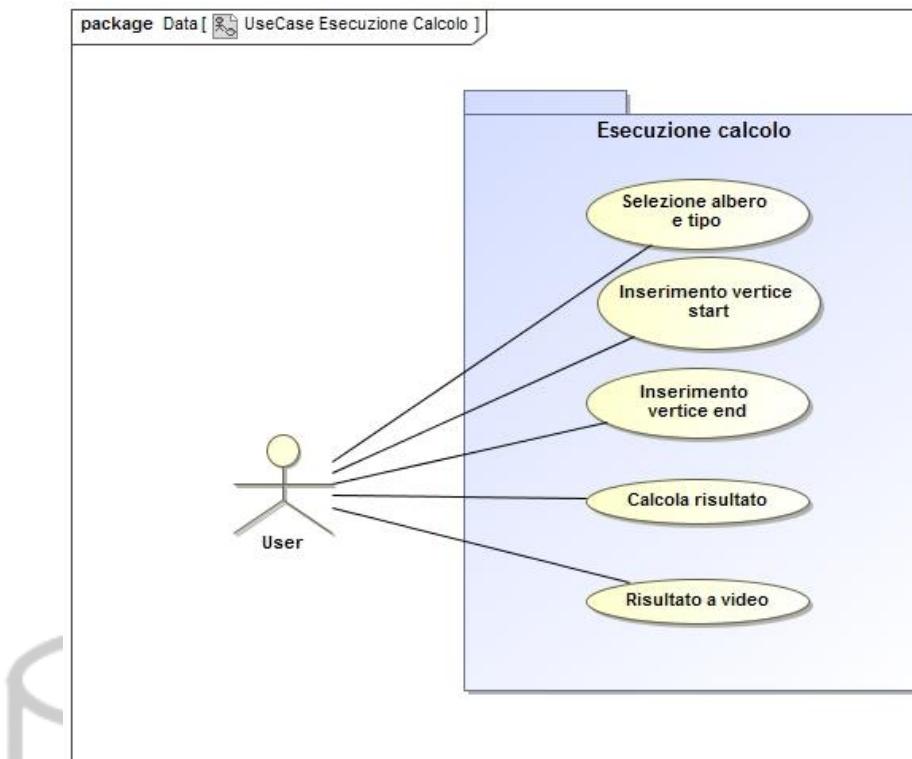
1. Creazione albero:

- Inserimento Nome (5): Consente all'utente di inserire il nome dell'Albero, che identificherà univocamente un'istanza di albero.
- Inserimento Tipo (4): Consente all'utente di inserire il tipo dell'Albero.
- Inserimento Split Size (5): Permette all'utente di definire l'altezza dell'Albero.
- Inserimento Depth (5): Permette all'utente di definire il numero esatto di archi uscenti da un nodo.
- Inserimento Vertex e Edge attribute list (5): Tramite checkbox, l'utente può scegliere gli attributi da inserire nei Vertici e nei Nodi dell'Albero. Le prime checkbox, obbligatoriamente spuntate, sono relative a Vertex e Edge, e fanno riferimento ad attributi di valore Intero su cui l'Engine eseguirà il calcolo, i restanti riferiscono ad attributi di tipo Stringa.
- Crea Albero (3): Chiamata al codice interno alla GUI che crea l'albero .XML effettivo.
- Selezione percorso salvataggio file (2): Consente all'utente di scegliere il percorso di salvataggio del file .XML.



2. Upload su database:

- Upload su database (3): Permette all'utente di selezionare dal FileSystem il file .XML contenente l'Albero da caricare sul DBMS (una volta premuto il tasto "Apri").



3. Esecuzione calcolo:

- Selezione albero (4): Permettono all'utente di selezionare in modo univoco dal DBMS l'albero su cui si vuole effettuare il calcolo filtrandolo tramite il Nome (Apparirà una lista di alberi precedentemente caricati sul database).
- Inserimento vertice Start ed End (5): L'utente seleziona i vertici di interesse su cui lavorare.
- Calcola risultato e Risultato a video (4): L'utente, cliccando il pulsante "Calcola", avvia l'engine esterno il quale, una volta eseguite le operazioni, restituirà il risultato alla GUI che lo stamperà a video.

DBMS:

- Nel DBMS è possibile caricare un albero già esistente oppure appena creato (4)
- Al nostro DBMS con i suoi attributi e le sue tabelle, devono essere passati valori del tipo previsto (5)
- Al DBMS, la GUI, dovrà passare valori che corrispondono al tipo degli attributi dell'entità (5)
- Il DBMS si aspetta dalla GUI i dati dell'albero, provenienti dal file XML. (5)

Engine:

- Il sistema deve garantire che i parametri passati all'engine siano compatibili (dello stesso tipo) con quelli su cui viene effettuato il calcolo. (5)
Es. Se il sistema effettua il calcolo su due o più int se passato uno string in ingresso rileva un errore.
- Il tipo del risultato che l'engine passerà alla GUI dovrà essere compatibile con quello che la GUI stessa si aspetta. (5)
Es. La GUI si aspetta un risultato in int, quindi non può ricevere uno string.
- Il sistema dovrà garantire un accesso concorrente. (4)
- L'engine dovrà essere richiamabile tramite GUI esterne. (5 – RICHIESTO DA CUSTOMER)

3.2 Scenari FR (Functional Requirements)

I requisiti funzionali nel nostro sistema sono espressi come caso d'uso (use case), ovvero di come gli attori possono raggiungere l'obiettivo attraverso il sistema.

Come ripetuto nelle assunzioni, l'analisi dei nostri scenari fa riferimento in modo principale alle nostre tre macro-funzioni ovvero la creazione dell'albero, l'upload del file sul database e l'esecuzione del calcolo che si occuperà di chiamare l'Engine.

Creazione albero:

Successo:



Fallimento:



Upload su database:

Successo:



Fallimento:

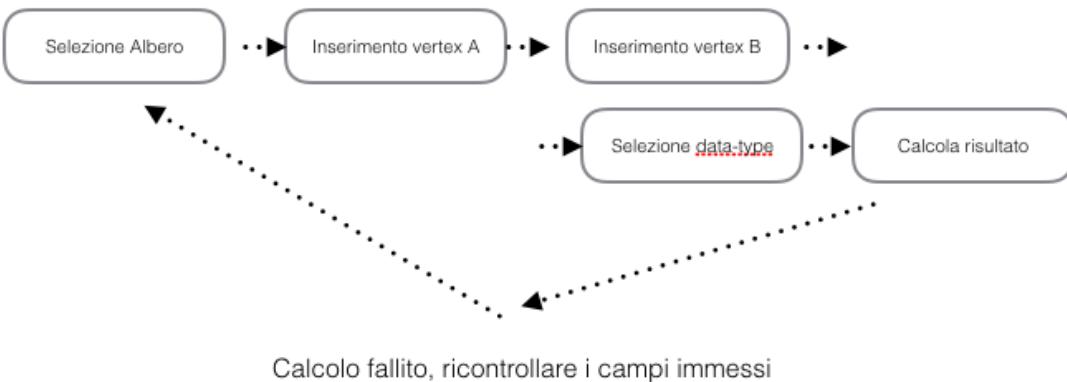


Esecuzione calcolo:

Successo:



Fallimento:



Descrizione use-case priorità alta

Use Case name	Inserimento split-size e depth
Participating actors	Utente
Descrizione	Permette all'utente di inserire i valori split-size e depth nella GUI
Evento scatenante	Utente autorizzato ad usare il sistema con a disposizione i dati dell'albero
Uses	File xml dell'albero creato correttamente

Use Case name	Selezione albero
Participating actors	Utente
Descrizione	Permette all'utente di selezionare il path name dell'albero da creare
Evento scatenante	Utente autorizzato ad usare il sistema con a disposizione la posizione dell'albero
Uses	Albero selezionato correttamente

Use Case name	Inserimento vertice start e end
Participating actors	Utente
Descrizione	L'utente può inserire il vertice di partenza e di destinazione
Evento scatenante	Utente autorizzato ad usare il sistema
Uses	Vertici inseriti correttamente

3.3 Non Functional Requirements

GUI:

- Usability: La GUI deve presentare un'interfaccia semplice e immediata.
- Reliability: La GUI deve creare l'albero seguendo precisamente i dati immessi dall'utente e deve interfacciarsi con il database e chiamare l'Engine senza errori.
- Availability: La GUI deve essere sempre disponibile nel momento in cui l'utente ne abbia bisogno (Tempo effettivo di funzionamento / Tempo Totale = 1).
- Performance: I tempi di attesa della GUI sono nell'ordine dei secondi, soprattutto durante l'esecuzione delle operazioni più pesanti (Serializzazione/Deserializzazione XML e Interfacciamento con il database).

DBMS:

- Performance: Il DBMS deve essere rapido nel restituire il risultato delle query
- Scalability: Il DBMS deve permettere di essere eseguito o consultato parallelamente da più persone

ENGINE:

- La **sicurezza** del nostro sistema è a carico dell'azienda costume.
- Il **calcolo sui vertici** dovrà essere svolto in un tempo ragionevole (qualche secondo al massimo).
- Il sistema dovrà essere in grado di svolgere operazioni di **calcolo** su alberi di grandezza pari a milioni di vertici.



SQLR

4.Content

Provenienza dati: interna al sistema.

API esterne: Database (MSSQL 2016).

5. Assumptions

Formato file: si assume che il file di creazione e sul quale lavoreremo, sia in formato xml.

Tipologia customer: si ipotizza che il sistema sia utilizzato da persone qualificate nel settore, competenti in materia.

Ottimizzazione sistema: Il programma dovrà fornire output in tempi accettabili, nè troppo lunghi e non necessariamente real time.

Sicurezza: assumiamo che il sistema sia in un ambiente sicuro per la protezione del prodotto.

Data type: assumiamo che il tipo dei vertici e degli archi dell'albero, sia assegnato in modo casuale, come d'accordo con il customer.

Scenari: si assume che oltre agli scenari efficienti in cui tutto vada per il verso giusto esistano scenari in cui qualcosa potrebbe fallire al nostro customer, nello specifico potrebbe fallire la creazione dell'albero, l'upload su database e la chiamata engine per motivi specifici.

Attributi: Valore degli attributi su cui fare il calcolo, RANGE vertici da 1 a 99, RANGE archi da 1 a 55 .

DBMS: Il database nel sistema deve essere MSSQL.

Engine: Se chiamato dalla GUI, l'Engine deve trovarsi nella stessa cartella, e chiamarsi 'Engine.exe' .

GUI: La GUI avrà estensione .EXE.

Engine: vista la struttura dati utilizzata, non sarà possibile fare calcoli su vertici che giacciono sullo stesso livello e non sarà possibile partire da figli per arrivare al padre (come da logica fornita dal customer.).

Vertex e Edge: vista la struttura dati utilizzata, si è preferito accoppare l'oggetto Edge dentro l'oggetto Vertex, visto che in un albero N-ARIO ogni vertice ha UN SOLO arco entrante.

DBMS: è stata fatta l'assunzione che la chiave univoca dell'albero sia il nome dello stesso per una questione di efficienza e ricerca.

Attributi: i due attributi principali, rispettivamente per Vertex e Edge, nonché i default saranno di tipo intero, mentre quelli opzionali a scelta dell'utente saranno di tipo string, indicando una tipologia di attributo. I selezionabili saranno 4 rispettivamente per le due classi in più a quello di default per un totale di 5 attributi per la class Vertex e 5 per la class Edge. L'utente potrà quindi selezionare il numero di attributi a suo piacimento (es.3-4-1).

6. Analysis Model

Il team ha prodotto il diagramma di Analysis Model affrontando diverse decisioni.

La prima riguarda l'utente, il quale dialoga con la “**GUI**”. La “**GUI**” nel nostro diagramma è un Boundary Object perché si interfaccia direttamente con il System Actor “**Utente**”. Da questo punto a seguire, ciascun Controller Object sarà tale poiché rappresenta la logica alla base dell'esecuzione delle funzioni del nostro sistema.

Il Controller Object “Creazione albero” utilizza l'Entità “**Albero**”. L’ “Albero” è un' Entity Object poiché rappresenta la struttura dati di un albero.

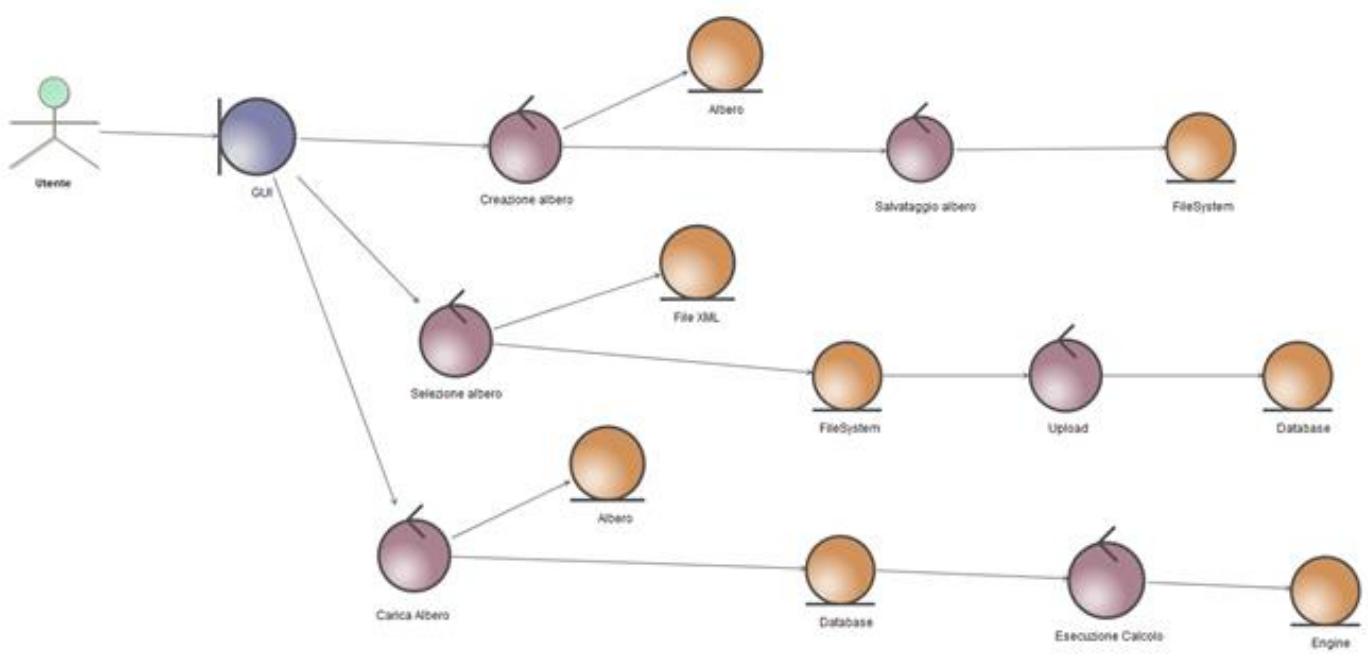
A seguire vi è il Controller Object “**Salvataggio albero**” che permette , interfacciandosi con il “**FileSystem**”, di generare un file di tipo XML. Il “**FileSystem**” nel nostro diagramma è un Entity Object poiché contiene i dati del sistema (File .XML) .

Il Controller Object successivo è il processo “Selezione albero”. Tale routine si interfaccia con l' Entity Object “**FileSystem**” al fine di trovare e caricare l'Entity Object “**File .XML**” che contiene le informazioni dell'albero. Il Controller Object che segue è l' “**Upload**”.

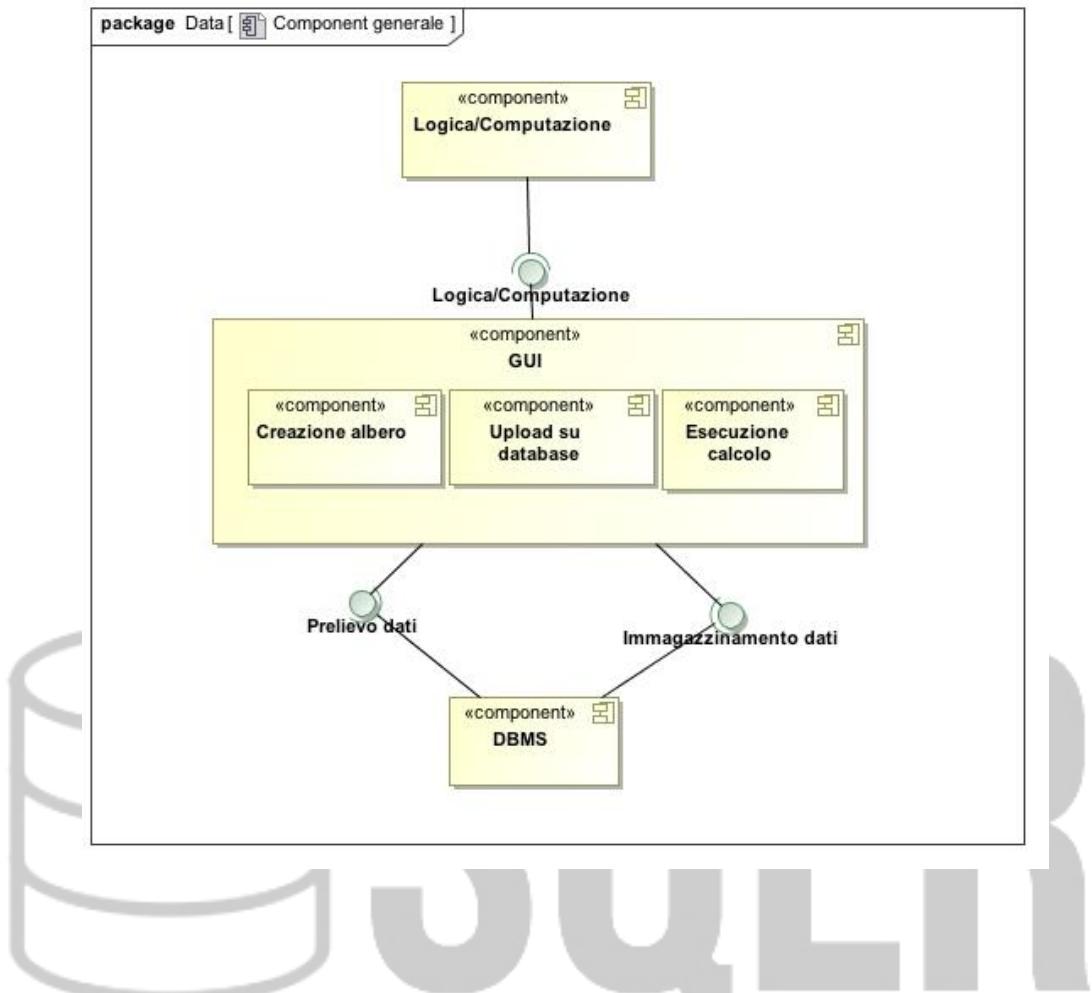
Tale funzione permette di caricare l'albero all'interno della base di dati dialogando con l' Entity Object “**Database**”, che quindi conterrà i dati di ogni “**Albero**” caricato .

Il Controller Object “**Carica albero**” comunica con l'Entità “**Database**” per importare l'Entity Object “**Albero**”.

Successivamente il Controller Object “**Esecuzione calcolo**” invoca l'Entity Object “**Engine**”, il quale svolge le operazioni di calcolo sull' albero precedentemente importato.



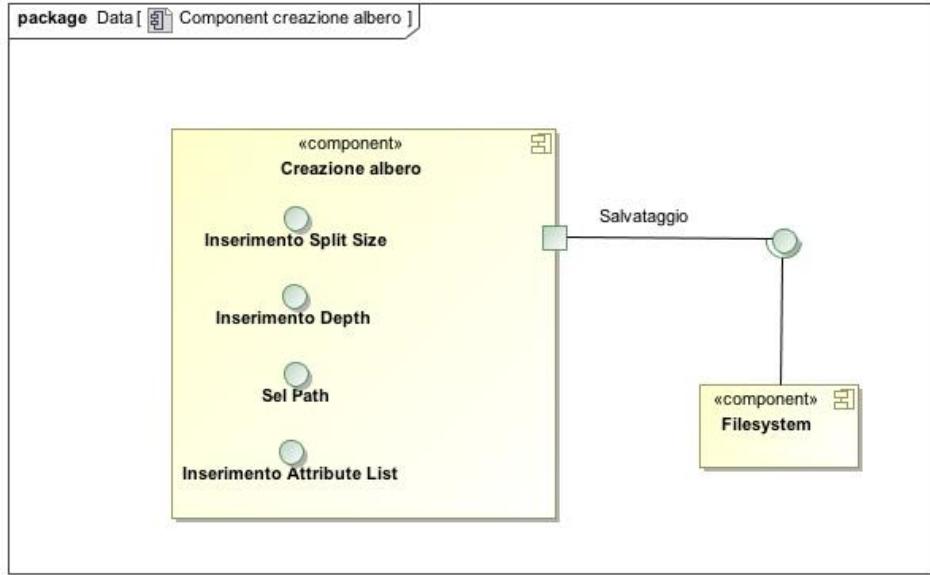
7. Component Diagram



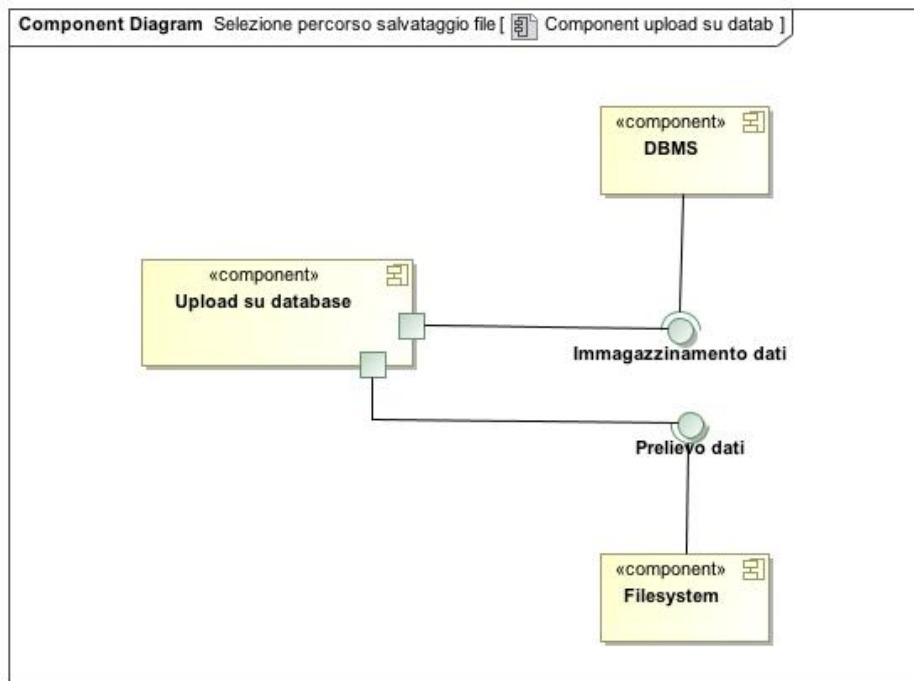
Nel nostro sistema sono state individuate le seguenti componenti:

- GUI
- Logica/Computazione
- DBMS

Il team ha deciso di suddividere la GUI in tre micro-componenti, rappresentati le tre macro-funzioni della stessa quali la creazione dell'albero, il relativo upload su DBMS e l'esecuzione del calcolo dei vertici (engine).

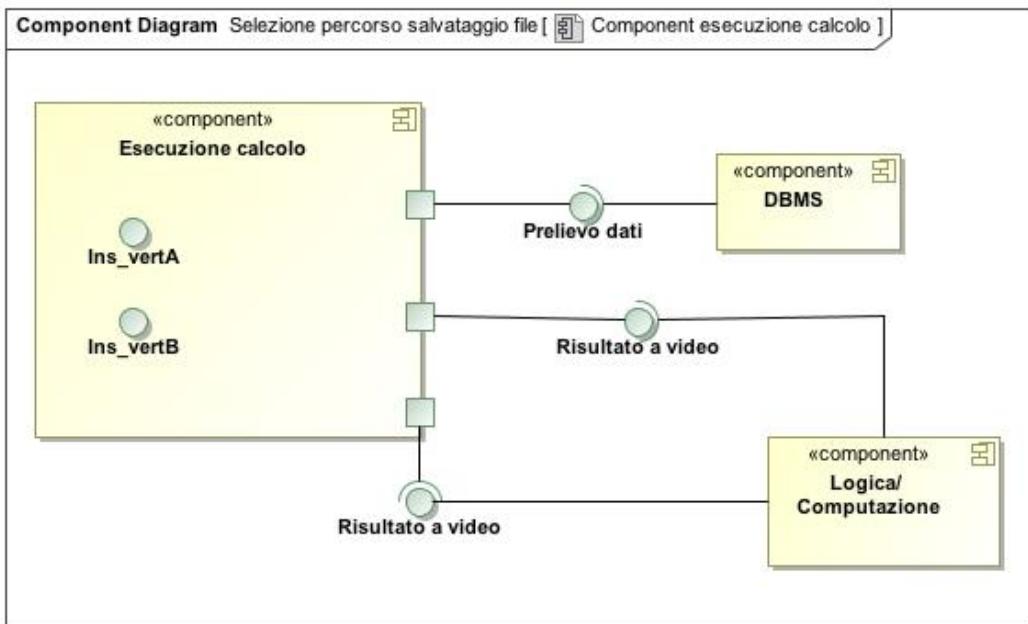


Il component diagram della componente Creazione albero è formata da interfacce corrispondenti alle funzioni di questa micro-componente, in particolare inserimento dati. In particolare vi è un'interfaccia con la componente Filesystem poiché la funzione di salvataggio del file dovrà necessariamente interagire con quest'ultimo per caricare il file su di esso.



La componente che fa riferimento all'Upload dovrà interfacciarsi alle componenti DBMS e Filesystem.

Alla prima per salvare il file albero selezionato tramite l'interfaccia "immagazzinamento dati" e la seconda per andare a prelevare sul filesystem l'albero creato in "creazione albero" tramite l'interfaccia "prelievo dati".



Infine la componente Esecuzione calcolo, che ricordiamo non è altro che l'ultima micro-componente della GUI analizzata inizialmente, si interfacerà con varie componenti differenti a seconda della funzione scelta ed avrà a sua volta interfacce interne.

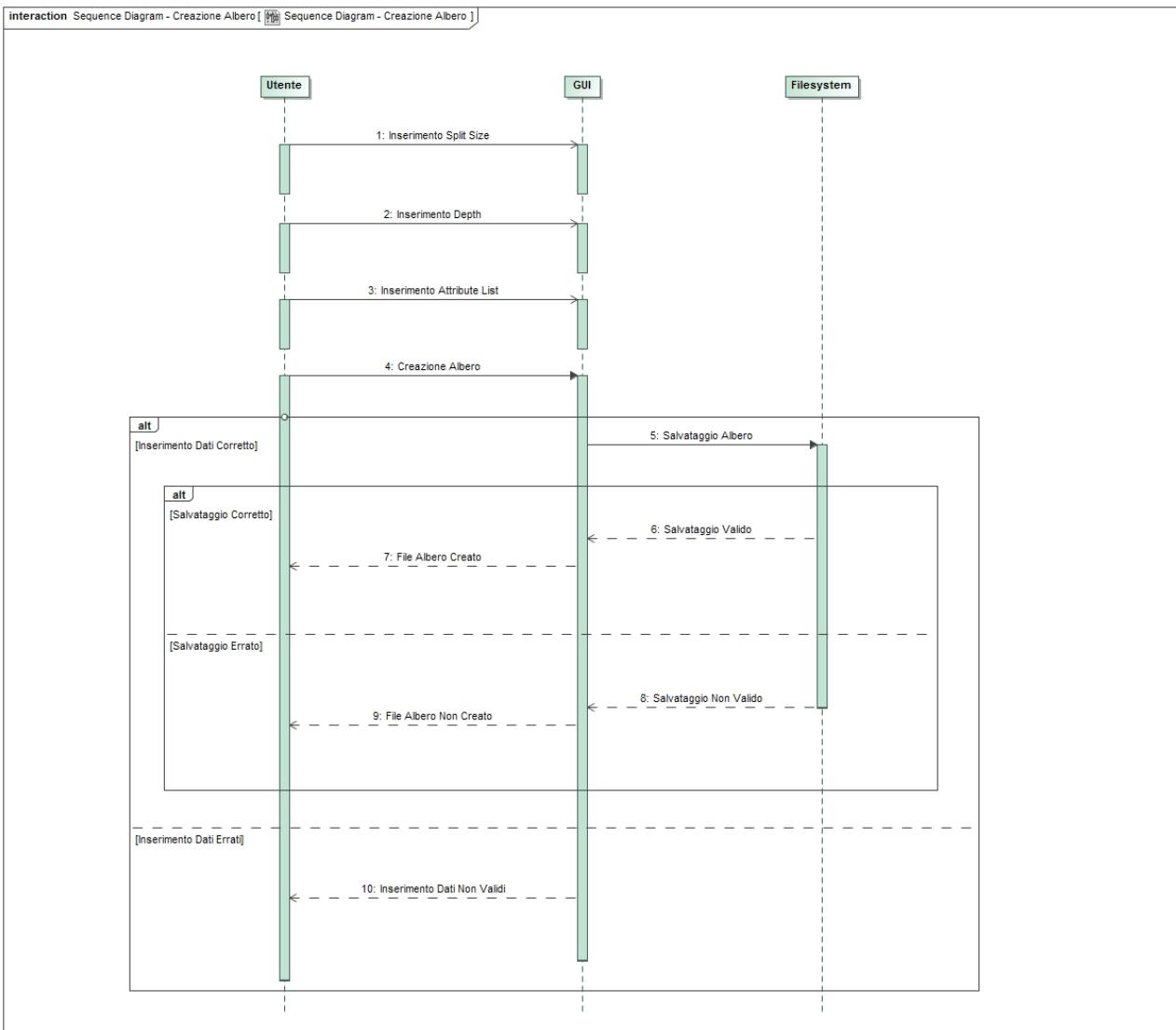
In particolare abbiamo l'interfaccia "prelievo dati" verso la componente componente DBMS che ci permetterà di prendere i dati relativi ad un albero, utili al nostro calcolo, caricati su database.

L'interfaccia "**PIPE_out**" permette alla GUI di passare all'engine l'oggetto albero con i relativi dati di calcolo mentre l'interfaccia "**PIPE_in**" farà il percorso inverso restituendogli il risultato che mostrerà a video; queste interfacce saranno ovviamente coinvolte con le componenti "**Esecuzione Calcolo**" e "**Logica/Computazione**".

Infine vi saranno le interfacce interne quali "**ins_vertA**" e "**ins_vertB**" che prenderanno in input graficamente dei valori numerici che verranno poi passati tramite l'interfaccia "**PIPE_out**" alla componente rappresentante l'engine.

8.Sequence Diagram

Il team ha deciso di generare tre Sequence Diagram, ciascuno corrispondente ad un macroscenario differente: quello della creazione di un nuovo albero, quello dell'upload su database ed, infine, quello dell'esecuzione del calcolo mediante l'engine esterno. Per ognuno di questi, verrà analizzato l'intero “flow of events”, così da mostrare sia i casi di successo che quelli di fallimento.



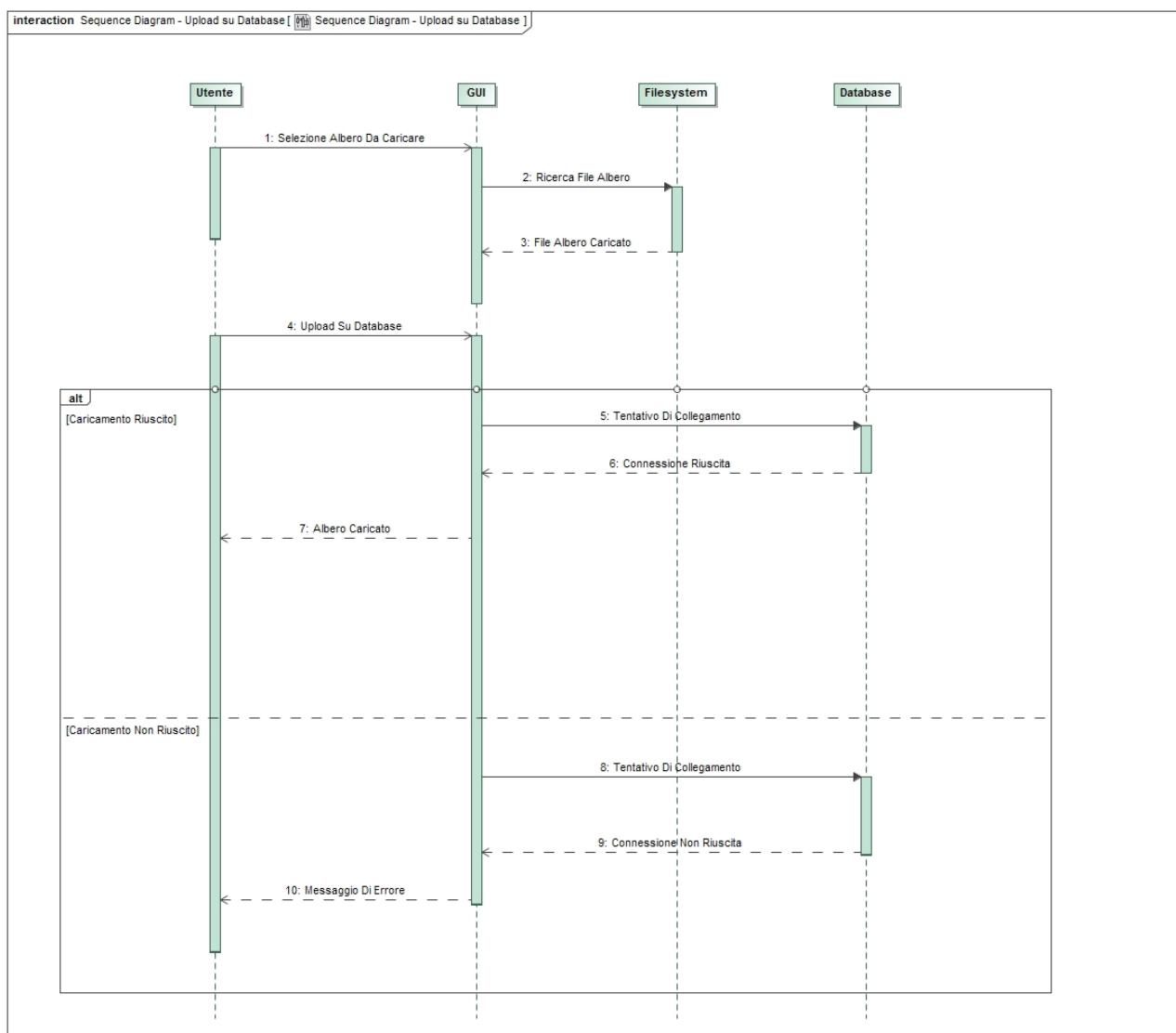
Il primo Sequence Diagram inserito corrisponde allo scenario di creazione di un nuovo albero, generato a partire da una serie di dati inseriti dall'utente, mediante l'utilizzo della GUI e del suo codice interno. In particolare si richiederà innanzitutto l'immissione dei valori di "split size" e "depth", che andranno a definire la dimensione totale dell'albero; in un secondo momento andrà definita l'"attribute list", che permetterà di assegnare degli attributi ai vertici e ai nodi dell'albero. Dopo aver inserito questi dati, l'utente potrà proseguire con la creazione dell'albero, scegliendo il percorso di salvataggio del file, interfacciandosi con il filesystem (componente necessaria ma esterna al nostro sistema).

Nel caso in cui tutto vada a buon fine, l'utente riceverà un messaggio di conferma e potrà trovare il file nel percorso che ha scelto. I principali casi di fallimento all'interno di questa procedura sono di due tipi:

- errore di datatype (“split size” di tipo float, quando invece è richiesto int), valori non validi o campi vuoti. In questo caso l'utente riceverà un messaggio di errore nel momento in cui selezionerà la voce “crea albero”;
- errore generico durante il salvataggio. In questo l'utente riceverà un messaggio di errore nel momento in cui selezionerà la voce “salva”, all'interno della form di salvataggio.

In entrambi i casi l'albero non verrà creato; l'utente potrà comunque modificare i dati o il percorso di salvataggio senza dover riavviare il programma.

In questo scenario ci saranno relazioni tra: utente e GUI; GUI e filesystem.



Il secondo Sequence Diagram riassume lo scenario di upload sul database di un albero precedentemente creato.

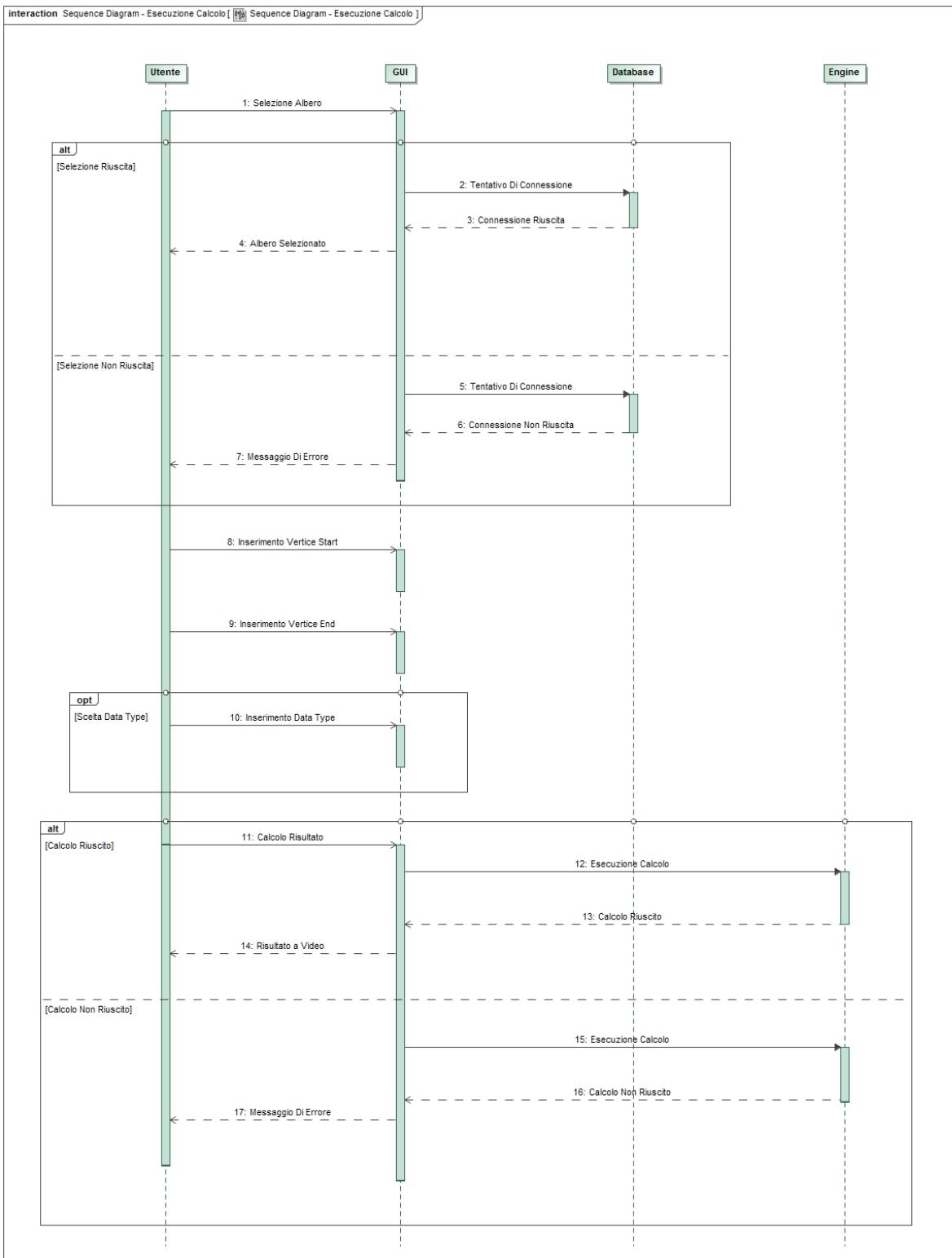
L'utente, cliccando sulla voce "seleziona albero" dovrà scegliere il file da caricare all'interno del filesystem; in seguito, selezionando "upload su database", l'utente attiverà la funzione di caricamento dell'albero. Questa procedura richiede che la GUI comunichi con il database.

In caso di successo, l'utente riceverà un messaggio di conferma e l'albero verrà caricato sul database.

Un eventuale errore potrebbe essere generato da un qualsiasi fallimento avvenuto durante la comunicazione tra GUI e database (timeout risposta, fallimento connessione, ecc.). In questo caso l'utente riceverà un messaggio di errore e dovrà ripetere la procedura di cariamento.

In questo scenario ci saranno relazioni tra: utente e GUI; GUI e filesystem; GUI e database.





Il terzo e ultimo Sequence Diagram descrive lo scenario di esecuzione del calcolo mediante l'engine esterno.

L'utente, sempre attraverso la GUI, seleziona dal database l'albero su cui vuole eseguire il calcolo. In caso di fallimento, egli riceverà a video un messaggio di errore, altrimenti potrà proseguire con l'operazione di calcolo. A questo punto, verrà richiesto di inserire il "vertice start" e il "vertice end". Questi due valori rappresentano i nodi iniziale e finale su cui eseguire il calcolo.

Opzionalmente, l'utente può anche inserire il tipo dei nodi sui cui calcolare la somma.

A questo punto, selezionando la voce “esegui calcolo”, la GUI chiamerà l’engine esterno passando come parametri l’albero caricato sul database, e i dati inseriti dall’utente. Si aprono due possibilità:

- calcolo riuscito: l’utente riceverà a video il risultato del calcolo;
- calcolo fallito: l’utente riceverà a video un messaggio di errore.

In questo scenario ci saranno relazioni tra: utente e GUI, GUI e database, GUI ed engine.

9. Class Diagram

1. Class Identification:

Class ALBERO:

Il team ha deciso di importare questa classe dal diagramma generale poiché ritenuta fondamentale al funzionamento dell’engine, in quanto quest’ultimo lavorerà proprio su essa. La classe albero è composta dall’oggetto albero il quale a sua volta è caratterizzato da quattro attributi (split-size; depth; attribute; type) quali specificano la struttura dell’albero in considerazione.

Class Engine:

La classe Engine è il cuore del nostro sistema, poiché contiene il codice che ci permetterà di effettuare il calcolo effettivo tra vertici. La classe contiene l’oggetto Engine che è caratterizzato da operazioni che indicano il processo obbligato per restituire un output corretto.

2. Find Attributes:

obj ALBERO:

Gli attributi di questo oggetto non saranno altro che le caratteristiche proprietarie dell’albero ovvero la split-size, la depth, gli attribute scelti dall’attribute list ed infine il type dell’albero stesso.

3. Find Method:

I metodi identificati sono tutti contenuti nell’oggetto Engine in quanto è l’unico oggetto del nostro Class Diagram ad effettuare operazioni.

La prima operazione è Import_datatree() che indica l’albero passato all’engine dall’esecuzione calcolo (che lo preleverà dal DBMS).

La seconda è l’Import_vertex() che non è altro che il passaggio dalla GUI (in particolare dell’esecuzione calcolo) all’engine dei due vertici (partenza e arrivo) che serviranno per calcolare il cammino minimo.

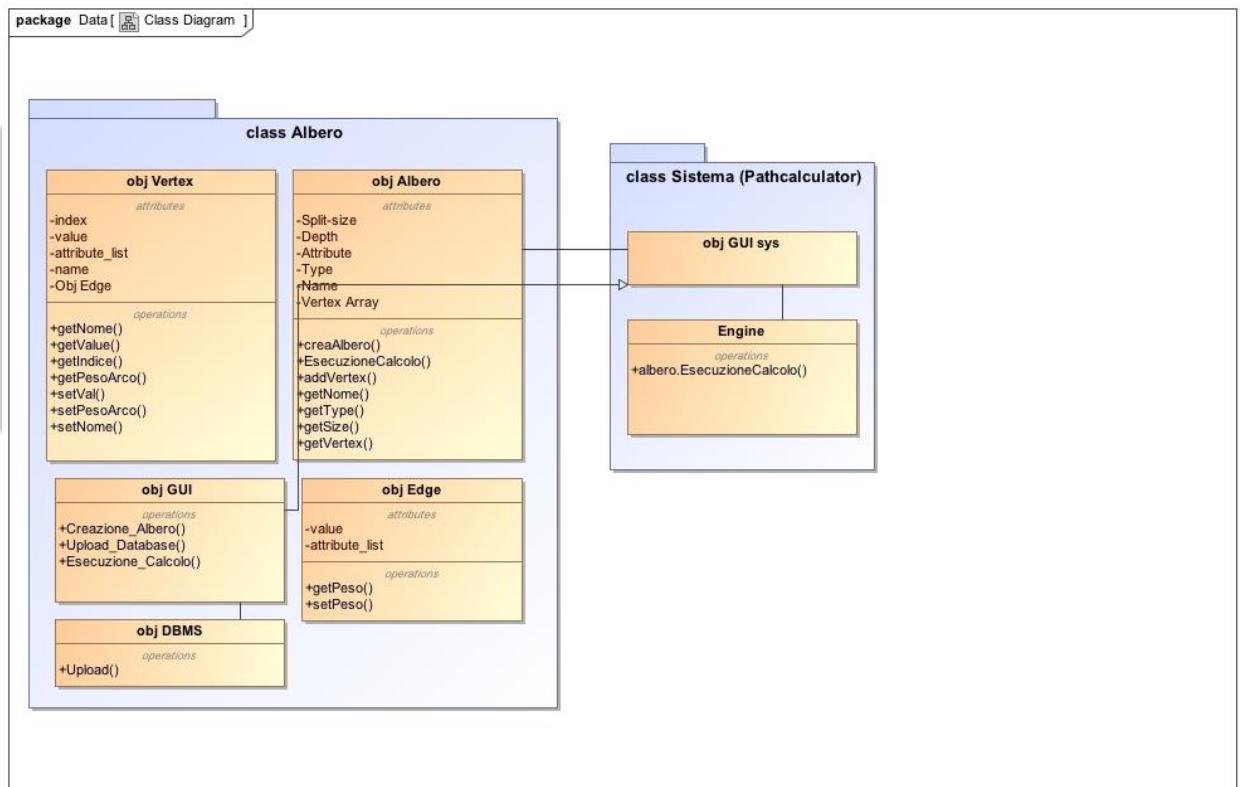
La terza (la fondamentale) è il code() che è il codice vero e proprio, nonché un’algoritmo che ottenuti i dati in ingresso (albero, vertex A, vertex B) si occuperà di calcolare il cammino minimo da A a B e di restituirci il risultato, cercando di ottenere le migliori prestazioni possibili.

L'ultima operazione è l'output() che non sarà altro che il passaggio inverso da Engine a GUI (in particolare dell'esecuzione calcolo) del risultato del calcolo, che quest'ultima metterà a video. Riguardo i metodi contenuti negli oggetti, essi fanno riferimento a frammenti di codice individuabili nella sezione Prototipo Engine.

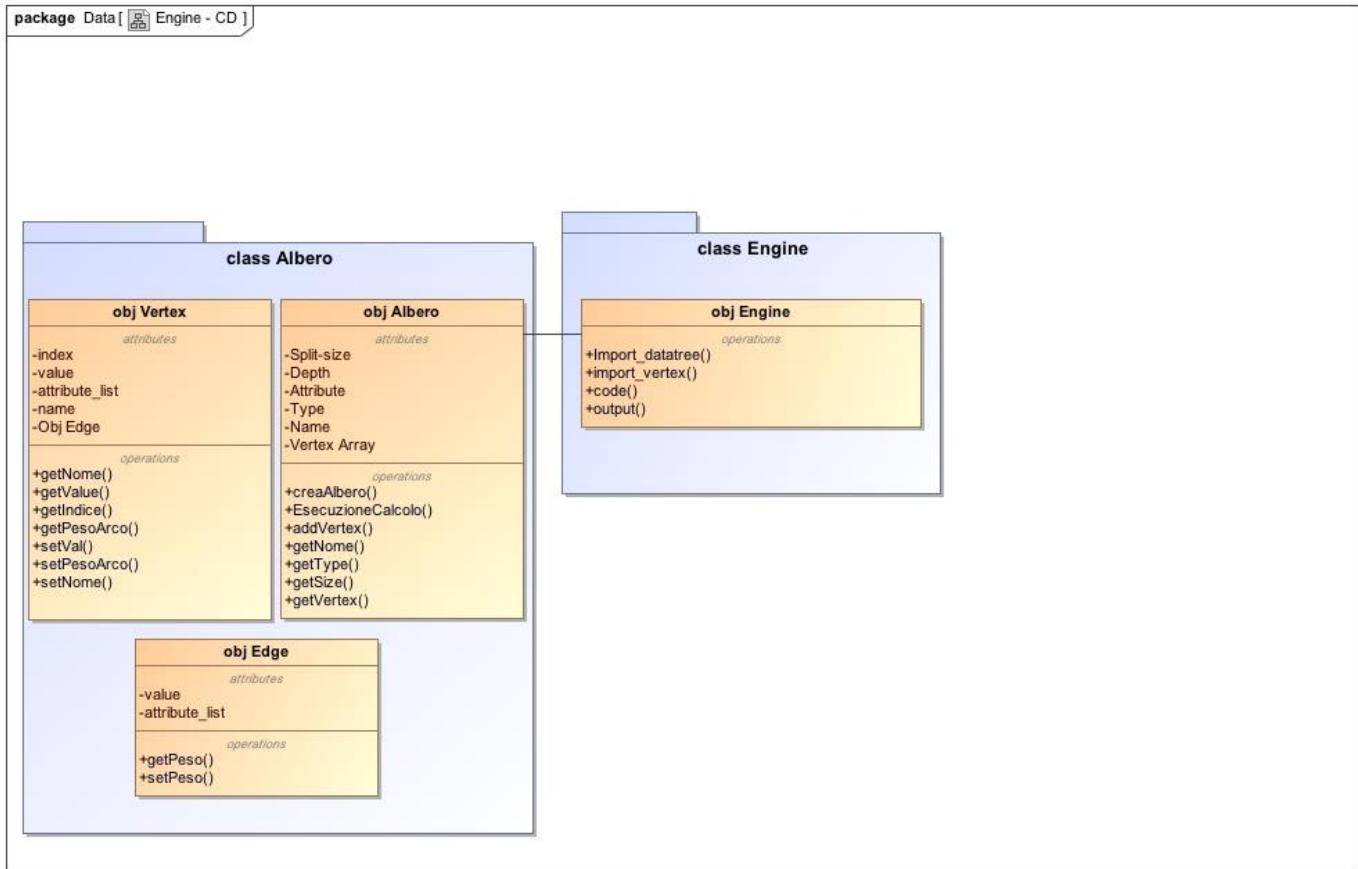
4. Find the associations between classes:

Le associazioni identificate e riportate nel diagramma consistono in quella tra l'oggetto Albero e l'oggetto Engine in quanto il primo è fondamentale ed imprescindibile per il secondo, nello specifico nell'operazione import_datatree() nella quale verrà importato l'albero stesso compreso dei relativi attributi.

Generale:



Engine:



10. Design Decision

Per definire le componenti del nostro sistema al fine di massimizzare i requisiti funzionali e NFR il team ha deciso di suddividere quest'ultimo in tre macro-componenti:

- Logica/Computazione
- GUI
- DBMS

In particolare in questa versione v2, la componente GUI verrà analizzata più in profondità per analizzare nello specifico le tre componenti.

1. Identificazione componenti

Logica/Computazione: è stata introdotta come componente in quanto corrisponde all'engine effettivo, quindi al file c# (o .exe) richiamabile dalla componente GUI o da GUI esterne proprietarie del customer.

GUI: introdotta come componente perché è il fulcro del nostro sistema, mediatore tra utente e codice. Permette infatti al customer di creare o apportare operazioni sull'albero tramite form di input e pulsanti di funzioni.

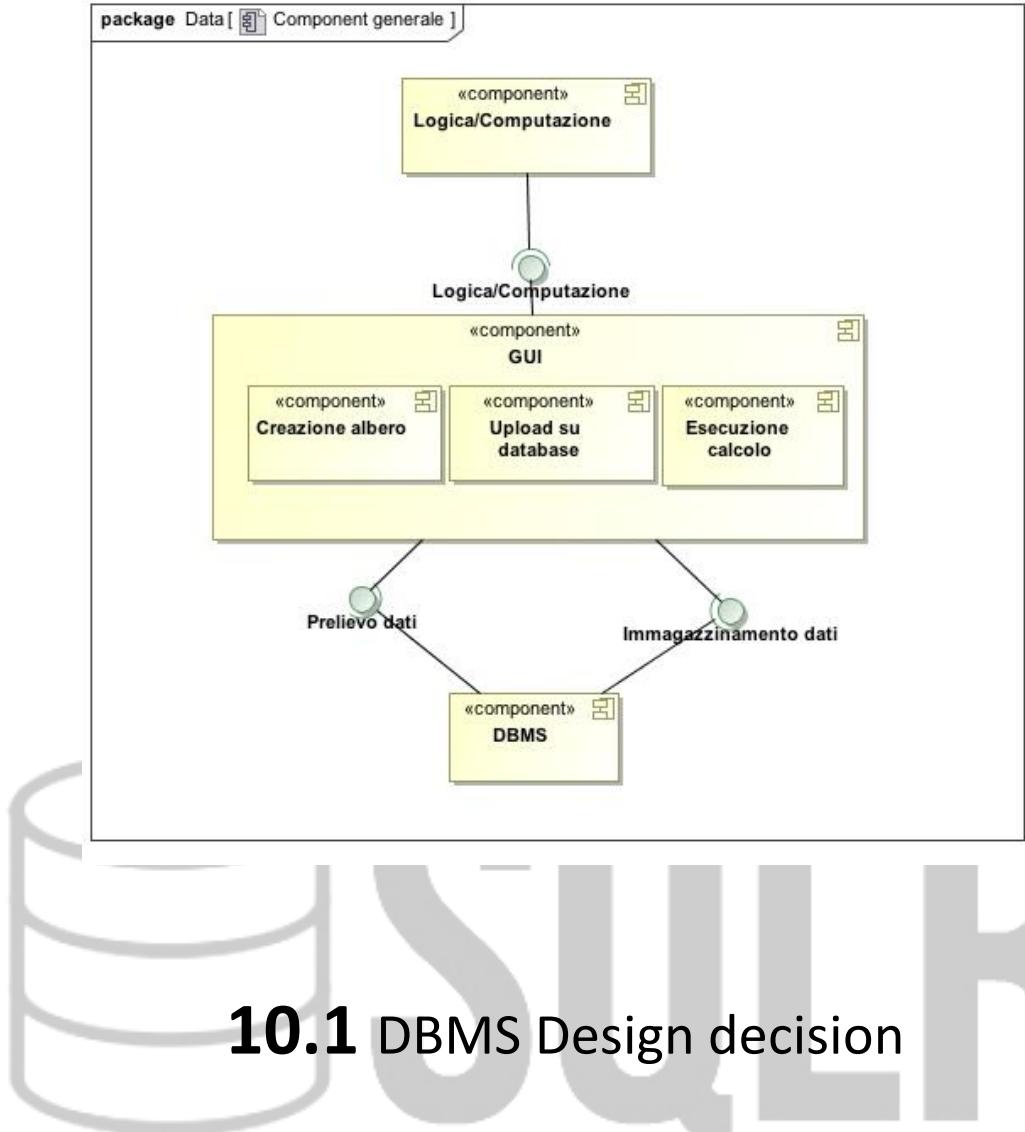
DBMS: componente fisica del nostro sistema, corrispondente al database sulla quale verranno caricati e prelevati i gli alberi con i rispettivi dati.

2. Identificazione interfacce

Logica/Computazione: mediatrice tra GUI e Engine, permette la chiamata del calcolo e restituisce alla GUI stessa il risultato di quest'ultimo.

Prelievo Dati: interfaccia tra GUI (in particolare Esecuzione calcolo) e DBMS che corrisponde alla funzione che preleva i dati dal database mettendoli a disposizione dell'utente e soprattutto del calcolo.

Immagazzinamento Dati: permette all'Upload Database (micro componente della GUI) di salvare i dati di un albero (e i relativi attributi) all'interno delle tabelle del database.



10.1 DBMS Design decision

Il team ha deciso di utilizzare un'entità “*Albero*” per rendere più veloce la visualizzazione dell’elenco degli alberi disponibili. In questo modo, abbiamo evitato che la GUI, nel momento della ricerca degli alberi, debba controllare tutte le istanze di un eventuale attributo “albero” all’interno delle entità “*Edge*” e “*Vertex*”.

All’interno dell’entità “*Albero*” è stato deciso di inserire: 2 attributi nei quali saranno salvati “*split size*” e “*depth*”, che serviranno per risalire al padre di un vertice al momento dell’esecuzione del calcolo; una variabile “*tipo*” in cui sarà inserita una stringa e un attributo “*nome*” che conterrà il nome dell’albero che dovrà essere univoco.

In tutte le entità del DBMS è stato deciso di utilizzare una chiave primaria di tipo intero con proprietà IDENTITY per l’autoincrement.

Il team ha deciso di evitare l’approccio della struttura a stella, per quanto riguarda l’entità “*AttrUsage*”, perché, come detto anche dal customer, si evita che l’attributo “*ObjUid*” diventi troppo grande e che possa rallentare la velocità del sistema; quindi utilizzeremo due entità separate, che abbiamo chiamato “*VertexAttrUsage*” e “*EdgeAttrUsage*” per rendere più lineare l’interazione tra GUI e DBMS.

11. How FR and NFR are satisfied by design

GUI:

Ciascun requisito funzionale della GUI corrisponderà all'atto pratico ad un bottone della stessa.

I non funzionali invece corrisponderanno al codice ottimizzato (nel caso delle performance e availability) .

La usability sarà soddisfatta da un interfaccia grafica pulita e semplice da usare.

DBMS:

I FR del DBMS sono soddisfatti tramite la configurazione di MSSQL 2016 e il relativo inserimento di tabelle necessarie, mentre i NFR come la performance saranno soddisfatti tramite lo spartamento e l'adattamento delle tabelle relativamente alla nostra struttura dati (Array) e alle nostre assunzioni sugli oggetti Vertex e Edge.

ENGINE:

I FR dell'engine come ad esempio il richiamo da GUI esterne sarà implementato via codice, nello specifico il nostro engine.exe sarà richiamabile sia da terminale passandogli i parametri richiesti sia collegabile a GUI esterne o non.

Dato che i parametri e risultati devono essere di tipo corrispondente verrà implementato tramite una sincronizzazione di codice tra GUI ed engine stesso.

I NFR come la sicurezza verranno implementati dal customer mentre il fatto che il risultato deve essere restituito in un tempo ragionevole e che operi su milioni di vertici verrà soddisfatto dalla struttura dati utilizzata, molto semplice e di rapida ricerca.

12. Effort Recording

(Espresso in ore)

GUI:

LOG TEAM GUI					
Riccardo A. Di Prinzi					
Valentino Di Giosaffatte					
Flavio Furia					
When (Month/Day)	Time spent	Partners (please report how many people)	Brief Description of the performed task	Category	Sub-Category
1 2	2	3	Revisione form Esecuzione Calcolo	Doing	Code Implementation
2 2	2	5	Collegamento effettivo con il DBMS	Doing	Code Implementation
5 2	2	3	Documentazione GUI e revisione finale	Doing	Final Review
8 2	3	3	Cura dell'aspetto grafico	Doing	Graphic design

Doing: 9

Effort: $9 \times 3 = \underline{27}$

DBMS:

LOG TEAM DATABASE					
Loris Fratarcangeli					
Luca Ferrari					
When	Time Spent	Partners	Brief Description of the performed task	Category	Sub-Category
1 2	2	2	Revisione schema E-R e modifiche	Doing	Requirement & modeling
5 2	1	2	Implementazione delle modifiche apportate	Doing	Code Implementation
8 2	1	2	Revisione finale	Doing	Final Review

Doing: 4

Effort: $4 \times 2 = \underline{8}$

ENGINE:

LOG TEAM ENGINE					
Patrizio Pezzilli					
Stefano Cortellessa					
When	Time spent	Partners	Brief Description of the performed task	Category	Sub-Category
5 2	2	2	Revisione e ristrutturazione Esecuzione Calcolo	Doing	Code Implementation
8 2	1	2	Documentazione Engine e revisione finale	Doing	Final Review

Doing: 3

Effort: $3 \times 2 = \underline{6}$

EFFORT TOTALE: $27+8+6=41$

13. GUI Specification + Code

(Codice GUI completo reperibile su
http://github.com/patriziopezzilli/SQRL/tree/master/SQRL_V4)

- Il team ha deciso, come richiesto dal customer, di sviluppare l'applicazione nel linguaggio C# utilizzando, come ambiente di sviluppo, il programma Microsoft Visual Studio. La GUI, come richiesto è stata progettata per interfacciarsi con un database MSSQL.
- Tutti i Form della nostra GUI hanno dimensioni fisse e non modificabili dall'utente. Ogni Form è un singolo processo che al momento dell'avvio blocca l'interazione dell'utente con il Form "padre". Alla chiusura di un Form si riattiva il controllo dell'utente con il Form precedente.

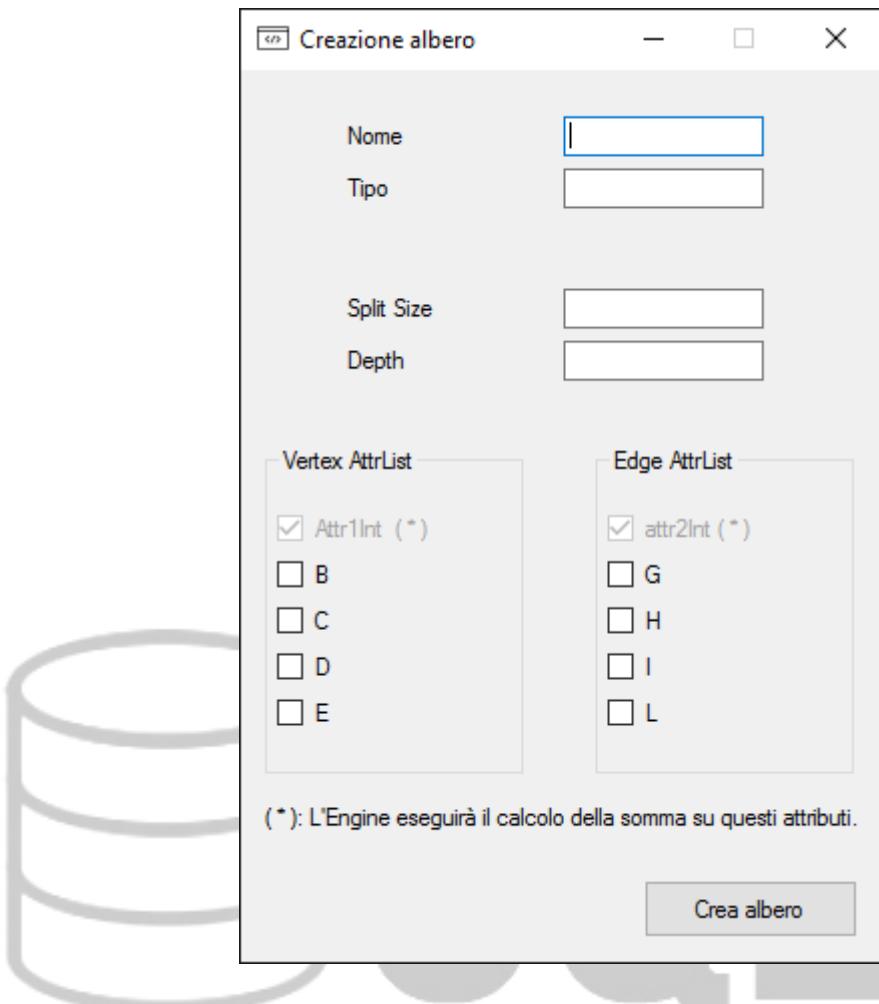
Da qui seguirà una descrizione dettagliata di ogni singolo Form:

PPC:



Permette di scegliere tra le 3 funzioni principali che rappresentano le macroaree del programma. La navigazione avviene mediante i buttons "Creazione Albero", "Upload su database" e "Esecuzione calcolo". La pressione di questi tasti aprirà le prossime 3 Form.

CREAZIONE ALBERO:

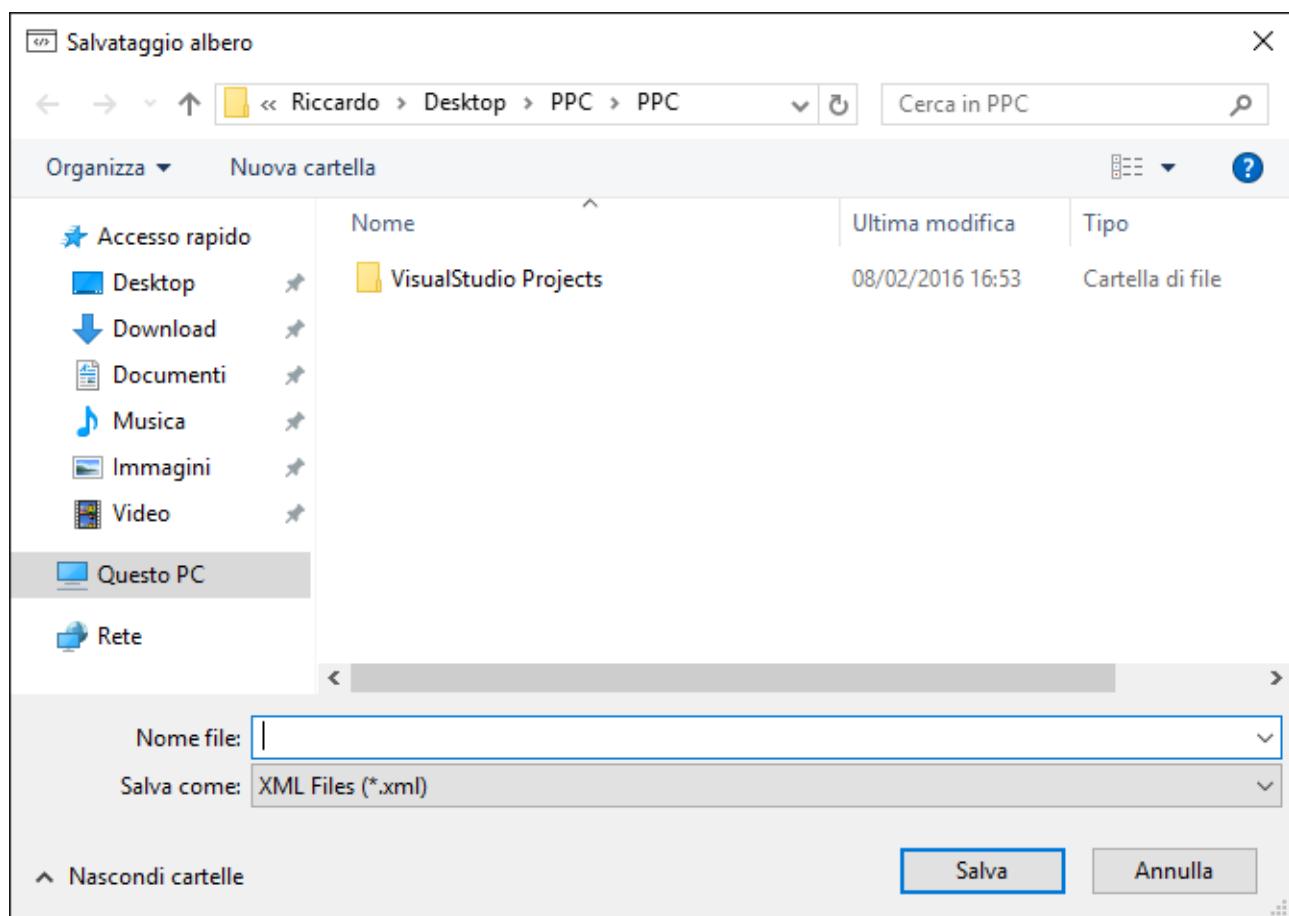


Questo Form permette la creazione di un oggetto della classe Albero (facente parte del codice interno della GUI e dell'Engine) con campi fissi e attributi a scelta dell'utente.

In questa finestra sono presenti 4 TextBox che permettono di definire le caratteristiche principali dell'Albero:

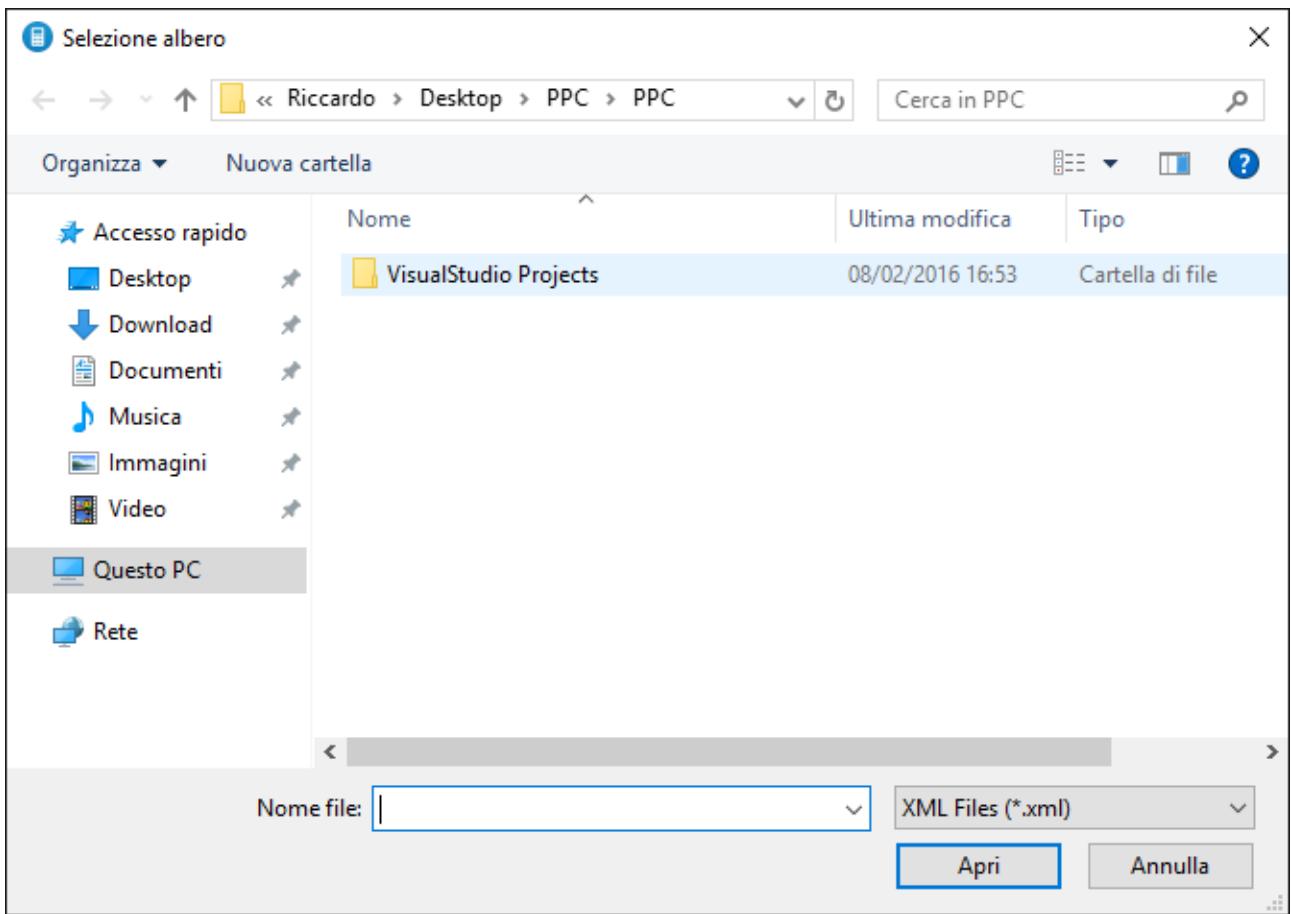
- Nome: Definisce il nome dell'Albero da creare. Necessita di un parametro di tipo stringa non vuota. Permette di identificare univocamente un albero nel database.
- Tipo: Definisce il tipo dell'Albero da creare. Necessita di un parametro di tipo stringa non vuota.
- Split size: Permette all'utente di scegliere l'altezza dell'albero da creare. Necessita di un parametro di tipo intero, compreso tra 1 (valore di default) ed un valore tale da non superare l'ordine dei milioni di vertici.
- Depth: Permette all'utente di definire il numero esatto di archi uscenti da un nodo. Necessita di un parametro di tipo intero, compreso tra 1 (valore di default) ed un valore tale da non superare l'ordine dei milioni di vertici.

Nel Form vi sono inoltre 2 GroupBox (Vertex AttrList e Edge AttrList) che contengono 5 CheckBox ciascuno. Ogni CheckBox consente all'utente di scegliere quanti attributi (di tipo stringa) inserire nell'albero. I primi attributi di ciascun GroupBox avranno valori casuali e sono obbligatori poiché l'Engine li utilizzerà per effettuare calcoli. I restanti conterranno una stringa alfanumerica randomica. Alla pressione del bottone "Crea albero" il programma risponderà in due modi possibili: stamperà un messaggio di errore a video se uno qualunque dei campi è stato riempito in modo non corretto, o una finestra di salvataggio SaveFileDialog che si occuperà di interfacciarsi con il FileSystem. Sul Form appena comparso è possibile scegliere il percorso di salvataggio e il nome del file (con estensione obbligatoria .XML) che conterrà l'albero serializzato.



Una volta che la serializzazione dell'albero viene completata, verrà stampato a video un messaggio di riuscita del compito e il Form "creazione albero" verrà chiuso.

UPLOAD SU DATABASE:

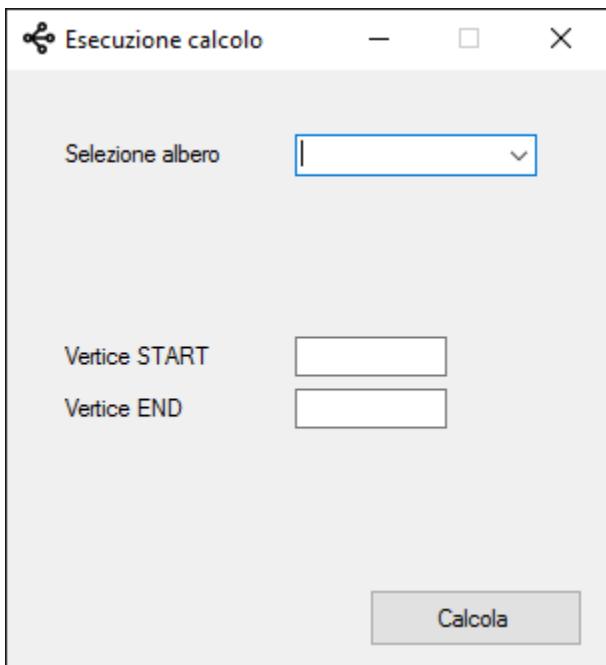


Il Form OpenFileDialog permette, interfacciandosi con il FileSystem, di scegliere un file (con estensione obbligatoria .XML) contenente l'albero.

Una volta scelto, il File .XML verrà analizzato e deserializzato, e l'oggetto albero recuperato verrà inserito sul database.

In caso di errore apparirà un messaggio a video che conterrà la sua descrizione. In caso di successo, sarà notificato all'utente un messaggio di riuscita del compito.

ESECUZIONE CALCOLO:



Questo Form ha il compito di eseguire calcoli su un albero tramite Engine esterno. In questa finestra è presente una ComboBox (menù a tendina), così definita:

- Selezione albero: Permette all’utente di scegliere uno tra gli alberi presenti nel database, identificati tramite il nome.

Sono presenti nel Form anche le 2 TextBox:

- Vertice START: l’utente inserisce qui il vertice iniziale da cui eseguire il calcolo. Se il Vertice START è maggiore del Vertice END, maggiore o uguale al numero di vertici dell’albero oppure se è minore di 0, verrà stampato a video un messaggio di errore.
- Vertice END: l’utente inserisce qui il vertice finale su cui terminare il calcolo. Se il Vertice END è minore del Vertice START, maggiore o uguale al numero di vertici dell’albero oppure se è minore o uguale a 0, verrà stampato a video un messaggio di errore.

Una volta inseriti i dati, cliccando il bottone “Calcola”, verrà eseguito un controllo sui dati immessi dall’utente. In caso di errore, esso sarà notificato all’utente, in caso di riuscita, verrà chiamato l’Engine esterno che eseguirà i calcoli sui i dati inseriti precedentemente dall’utente. Al termine dei calcoli, l’Engine restituirà il risultato alla GUI, che lo stamperà a video per l’utente, e il Form “Esecuzione calcolo” verrà chiuso.

Inserire un Vertice START che non sia antenato del relativo Vertice END, o viceversa, inserire un Vertice END che non sia discendente del relativo Vertice START non produrrà un messaggio di errore. Verrà intrapresa l’esecuzione del calcolo, senza

però giungere ad una conclusione: non verrà, dunque, restituito nessun risultato a video. Questo accade poiché, come da assunzione, è possibile eseguire il calcolo solo da un vertice fino ad uno dei suoi discendenti.

Code

(Solo delle funzioni rilevanti, codice completo reperibile su GitHub)

Creazione albero:

```
...p\PPC\VisualStudio Projects\GUI\GUI\Creazione_albero.cs 3
    if (checkBox9.Checked)
    {
        check[4] = true;
        chiamato[count++] = 4;
    }

    if (checkBox8.Checked)
    {
        check[5] = true;
        chiamato[count++] = 5;
    }

    if (checkBox7.Checked)
    {
        check[6] = true;
        chiamato[count++] = 6;
    }

    if (checkBox6.Checked)
    {
        check[7] = true;
        chiamato[count++] = 7;
    }

    //parametri corretti, determinazione checkbox scelti dall'utente
    //terminata
    //creazione istanza e salvataggio

    Tree albero = new Tree(nome, Tipo, splitSize, depth);

    albero.chiamato = chiamato;

    albero.creaAlbero(); //crea fisicamente l'albero

    //
    //lavori su albero finiti, ora salvataggio l'albero in file .xml
    //

    saveFileDialog1.ShowDialog();

    //Se il nome del file da salvare non è una stringa vuota allora lo
    //salvo
    if (saveFileDialog1.FileName != "")
    {
        //Serializzazione
        var writer = new System.Xml.Serialization.XmlSerializer(typeof
        (Tree));
        var wfile = new System.IO.StreamWriter(saveFileDialog1.FileName);
        writer.Serialize(wfile, albero);
        wfile.Close();

        //Notifica salvataggio completato
        MessageBox.Show("Salvataggio file completato", "GUI",
        MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
}
```

```
//chiusura Form creazione albero
this.Dispose();
}

}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
}

private void checkBox10_CheckedChanged(object sender, EventArgs e)
{
}
}
```



Esecuzione Calcolo:

```
...PPC\VisualStudio Projects\GUI\GUI\Esecuzione_calcolo.cs 3
//Controllo RANGE

if(!(startVertex >= 0 && startVertex < endVertex && startVertex <= albero.numNodi))
{
    //Range startVertex sbagliato!
    MessageBox.Show("Sono stati inseriti parametri non corretti", "GUI", MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

if(!(endVertex > 0 && endVertex > startVertex && endVertex <= albero.numNodi))
{
    //Range endVertex sbagliato!
    MessageBox.Show("Sono stati inseriti parametri non corretti", "GUI", MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

//Verifica andata a buon fine, dati OK

//scelta utente: Eseguire il calcolo?
var result = MessageBox.Show("Eseguire il calcolo?", "GUI", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

//scelta negativa da parte dell'utente
if (result == System.Windows.Forms.DialogResult.No) { return; }

//
//esecuzione GUI-Engine
//

Process myProcess = new Process();

myProcess.StartInfo.UseShellExecute = false;          //Attivazione
Engine
myProcess.StartInfo.FileName = "Engine.exe";           //Percorso
file da avviare
myProcess.StartInfo.Arguments = "GUI";                //Argomento
"GUI" da passare all'Engine per far capire ad esso che è stato
chiamato dalla nostra GUI
myProcess.StartInfo.CreateNoWindow = true;             //false =
finestra visibile, true il contrario

myProcess.Start(); //Processo Engine attivato

//
//comunicazione con Engine.exe
//
```

```
//Comunicazione con Engine.exe tramite Named Pipes
NamedPipeClientStream Pipe1 = new NamedPipeClientStream("Pipe1");
Thread.Sleep(1000);           //aspetta 1 secondo per far si che    ↵
    l'Engine sia pronto a ricevere la connessione dalla GUI
Pipe1.Connect();            //stabilisce connessione con Engine

//riempimento struttura messaggio
messaggio message = new messaggio();
message.Albero = albero;
message.startVertex = int.Parse(this.textBox1.Text);
message.endVertex = int.Parse(this.textBox2.Text);

message.Risposta = "";

//Invio messaggio a Engine
var f = new System.Xml.Serialization.XmlSerializer(typeof
    (messaggio));
f.Serialize(Pipe1, message); //mando il messaggio sulla pipe

//chiusura pipe invio
Pipe1.Close();

//Qui l'Engine esegue il calcolo e ci spedisce la risposta tramite ↵
Pipe2

//ricezione messaggio creando pipe di ricezione
NamedPipeServerStream Pipe2 = new NamedPipeServerStream("Pipe2");
Pipe2.WaitForConnection();

//Ricezione messaggio da Engine
messaggio ricevuto = new messaggio();

ricevuto = (messaggio)f.Deserialize(Pipe2);

//chiusura pipe ricezione
Pipe2.Close();

//compito riuscito

//Stampa risultato
MessageBox.Show("RESULT: " + ricevuto.Risposta, "GUI",
    MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

//chiusura Form esecuzione calcolo
this.Dispose();

}

catch (Exception ex)
{
    //accade eccezione, fallimento compito
    if (ex is System.ComponentModel.Win32Exception) MessageBox.Show
        ("ERRORE: " + "File 'Engine.exe' non trovato: assicurati che il
        file si trovi nella stessa cartella della GUI e si chiami
        " + ex.Message);
}
```

```
        'Engine.exe' "); //sollevata eccezione per Engine mancante
    else
        MessageBox.Show("ERRORE: " + ex.Message); //Tutte le altre
        // possibili eccezioni
    }
}

private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    int numVertex = 0; //contatore per il numero di vertici
    //query in DB per l'END Vertex
    try
    {

        SqlConnection conn = new SqlConnection("Persist Security
Info=False;Integrated Security=true;Initial Catalog=DB;server=
(local)");
        string s = string.Format("SELECT * FROM Albero, Vertex WHERE
Albero.Nome='{0}' AND Vertex.IdAlbero=Albero.Id ",
        comboBox1.Text);
        conn.Open();
        SqlCommand msc = new SqlCommand(s, conn);
        SqlDataReader msdr = msc.ExecuteReader();

        while (msdr.Read())
        {
            numVertex++; //contatore numero vertici
        }
        msdr.Close();
        conn.Close();

        //setto le textbox Vertex START e Vertex END
        textBox1.Text = string.Format("{0}", 0); //Vertex START
        textBox2.Text = string.Format("{0}", (numVertex-1)); //Vertex END
    }
    catch (Exception x)
    {
        MessageBox.Show(x.Message);
    }
}

public Tree riceviDB(string scelta)
{
    Tree albero = new Tree(); //albero vuoto da riempire

    //Setto connessione a database per ripescaggio dati
    SqlConnection myconn = new SqlConnection("Persist Security
Info=False;Integrated Security=true;Initial Catalog=DB;server=
```

```
(local)");

SqlCommand cmd = null; //il comand o di select
SqlDataReader myReader = null; //il lettore

//Tentativo di connessione al database
myconn.Open();
//connessione stabilita

//

//ripesco nomealbero, tipo, split size, depth
//

cmd = new SqlCommand(string.Format("SELECT * FROM Albero WHERE
    Nome='{0}'", scelta), myconn); //COMANDO SQL

//setto un datareader sul comando appena eseguito
myReader = cmd.ExecuteReader();

//Lettura dati dal database
if (myReader.Read()) //una sola volta
{
    albero.name = myReader.GetString(1);
    albero.type = myReader.GetString(2);
    albero.splitSize = myReader.GetInt32(3);
    albero.depth = myReader.GetInt32(4);

}

// Chiude il DataReader
myReader.Close();

//

//lettura di numNodi
//

long numVertex = 0;
cmd = new SqlCommand(string.Format("SELECT * FROM Albero, Vertex WHERE
    Albero.Nome='{0}' AND Vertex.IdAlbero=Albero.Id ", scelta),
    myconn); //COMANDO SQL

//setto un datareader sul comando appena eseguito
myReader = cmd.ExecuteReader();

//Lettura dati dal database per scoprire lunghezza albero
while (myReader.Read())
{
    numVertex++; //se leggo righe significa che ho vertici
}

//ho scoperto il numero di vertici
```

```
//creo l'albero di lunghezza giusta
albero.albero = new Vertex[numVertex];
albero.numNodi = numVertex;

// Chiude il DataReader
myReader.Close();

// lettura dei vertex
//

cmd = new SqlCommand(string.Format("SELECT Vertex.Name FROM Albero,
    Vertex WHERE Albero.Nome='{0}' AND Vertex.IdAlbero=Albero.Id ",
    scelta), myconn); //COMANDO SQL

//setto un datareader sul comando appena eseguito
myReader = cmd.ExecuteReader();

int count = 0;
//Lettura dati dal database
while(myReader.Read())
{
    albero.addVertex(new Vertex(count,count),count);
    albero.albero[count].nome = myReader.GetString(0);
    count++;
}

//Lettura dei vertex completata

// Chiude il DataReader
myReader.Close();

// lettura degli Edge
//

cmd = new SqlCommand(string.Format("SELECT Edge.Valore FROM Albero,
    Edge WHERE Albero.Nome='{0}' AND Edge.IdAlbero=Albero.Id ", scelta),
    myconn); //COMANDO SQL

//setto un datareader sul comando appena eseguito
myReader = cmd.ExecuteReader();

count = 0;
//Lettura dati dal database
while (myReader.Read())
{
    albero.albero[count].arcoentrante.val = myReader.GetInt32(0);
    count++;
}

//Lettura degli Edge completata
```

```
// Chiude il DataReader
myReader.Close();

// 
//Riempio attributi di Vertex
//

cmd = new SqlCommand(string.Format("SELECT AttrDef.Name, Value FROM      ↪
AttrDef, VertexAttrUsage, Vertex WHERE AttrDef.NomeAlbero='{0}' AND      ↪
AttrDef.AttrDefUid=VertexAttrUsage.AttrDefUid AND      ↪
Vertex.VertexUid=VertexAttrUsage.ObjectVUid", scelta), myconn); //      ↪
COMANDO SQL

//setto un datareader sul comando appena eseguito
myReader = cmd.ExecuteReader();

//Qui si prendono i dati giusti dal DB
count = 0;
long contatore = numVertex;
//Lettura dati dal database
while (myReader.Read())
{
    if ((myReader.GetString(0)).Equals("B"))
    {
        if (count == albero.albero.Length) count--; //Serve a      ↪
        correggere l'errore che succede all'importazione dell'ultima      ↪
        riga dal DB, che fa finire OutOfBound l'array di Vertex      ↪
        albero.albero[count].B = myReader.GetString(1);      ↪

    }

    if ((myReader.GetString(0)).Equals("C"))
    {
        if (count == albero.albero.Length) count--;
        albero.albero[count].C = myReader.GetString(1);
    }

    if ((myReader.GetString(0)).Equals("D"))
    {
        if (count == albero.albero.Length) count--;
        albero.albero[count].D = myReader.GetString(1);
    }

    if ((myReader.GetString(0)).Equals("E"))
    {
        if (count == albero.albero.Length) count--;
        albero.albero[count].E = myReader.GetString(1);
    }

    if ((myReader.GetString(0)).Equals("attr1Int"))
    {
        albero.albero[count].attr1Int = long.Parse(myReader.GetString(1));
    }
}
```

```
        (1));  
  
        contatore--;  
        count++;  
    }  
  
}  
  
//Lettura completata  
  
// Chiude il DataReader  
myReader.Close();  
  
  
//  
//Riempio attributi di Edge  
//  
  
cmd = new SqlCommand(string.Format("SELECT AttrDef.Name, Value FROM  
AttrDef, EdgeAttrUsage, Edge WHERE AttrDef.NomeAlbero='{0}' AND  
AttrDef.AttrDefUid=EdgeAttrUsage.AttrDefUid AND  
Edge.EdgeUid=EdgeAttrUsage.ObjectEUid", scelta), myconn); //COMANDO  
SQL  
  
//setto un datareader sul comando appena eseguito  
myReader = cmd.ExecuteReader();  
  
count = 0;  
  
contatore = numVertex;  
//Lettura dati dal database  
while (myReader.Read())  
{  
  
    if ((myReader.GetString(0)).Equals("G"))  
    {  
        if (count == albero.albero.Length) count--;  
        albero.albero[count].arcoentrante.G = myReader.GetString(1);  
    }  
  
    if ((myReader.GetString(0)).Equals("H"))  
    {  
        if (count == albero.albero.Length) count--;  
        albero.albero[count].arcoentrante.H = myReader.GetString(1);  
    }  
  
    if ((myReader.GetString(0)).Equals("I"))  
    {  
        if (count == albero.albero.Length) count--;  
        albero.albero[count].arcoentrante.I = myReader.GetString(1);  
    }  
  
    if ((myReader.GetString(0)).Equals("L"))  
    {
```

```
        if (count == albero.albero.Length) count--;
        albero.albero[count].arcoentrante.L = myReader.GetString(1);
    }

    if ((myReader.GetString(0)).Equals("attr2Int"))
    {
        albero.albero[count].arcoentrante.attr2Int = long.Parse
            (myReader.GetString(1));

        count++;
        contatore--;
    }

}

//Lettura completata

// Chiude il DataReader
myReader.Close();

//L'albero è stato ricreato

// Chiusura connessione database
myconn.Close();

//ritorno albero ricreato
return albero;

}
}
```

Form:

```
...ardo\Desktop\PPC\VisualStudio Projects\GUI\GUI\Form1.cs 1
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
//Serializzazione
using System.Xml.Serialization;
//SQL
using System.Data.SqlClient;

namespace GUI
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            Creazione_albero a = new Creazione_albero();
            a.ShowDialog();
        }

        private void button2_Click(object sender, EventArgs e) //pulsante Update su Database
        {
            Tree classe = null; //classe sarà l'istanza albero da ricreare dal file, ora è a null

            if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                //Deserializzazione
                System.Xml.Serialization.XmlSerializer reader = new System.Xml.Serialization.XmlSerializer(typeof(Tree));
                System.IO.StreamReader file = new System.IO.StreamReader(openFileDialog1.FileName);

                classe = (Tree)reader.Deserialize(file);
                file.Close();
            }

            if (classe == null) return; //non mi connetto al database se non ho importato nessun file .xml
        }
    }
}
```

```
//Update su Database a partire da albero deserializzato
updateDB(classe);

    //Compito riuscito
}

private void button3_Click(object sender, EventArgs e)
{
    Esecuzione_calcolo b = new Esecuzione_calcolo();
    b.ShowDialog();
}

private void openFileDialog1_FileOk(object sender, CancelEventArgs e)
{
}

// 
// 
// 
//FUNZIONE IMPORT DATABASE
// 
// 
// 

public void updateDB(Tree albero)
{
    try
    {

        string comando = null;

        //Apertura connessione con Database "DB"
        SqlConnection myconn = new SqlConnection("Persist Security
Info=False;Integrated Security=true;Initial Catalog=DB;server=
(local)");                                     ↵
                                                ↵

        //
        //inseriamo nome albero tipo, splitsize e depth
        //

        comando = string.Format("INSERT INTO Albero(Nome, Tipo, Split,
Depth) VALUES('{0}', '{1}', '{2}', '{3}')", albero.name,
albero.type, albero.splitSize, albero.depth);           ↵
                                                ↵

        //settiamo connessione
        SqlCommand cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
        cmd.Connection = myconn;

        //connessione
        myconn.Open();
```

```
//Query
cmd.ExecuteNonQuery();
//query su albero completata

// 
//occupiamoci degli attributi obbligatori di Vertex ed Edge
//

comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) ??
VALUES('{0}', '{1}')", "attr1Int", albero.getNome());
//settiamo connessione
cmd = new SqlCommand();
cmd.CommandType = System.Data.CommandType.Text;
cmd.CommandText = comando;
cmd.Connection = myconn;

cmd.ExecuteNonQuery();
//query completata

comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) ??
VALUES('{0}', '{1}')", "attr2Int", albero.getNome());
//settiamo connessione
cmd = new SqlCommand();
cmd.CommandType = System.Data.CommandType.Text;
cmd.CommandText = comando;
cmd.Connection = myconn;

cmd.ExecuteNonQuery();
//query completata

// 
//Riempimento restanti attributi se e solo se sono stati scelti
//dall'utente
//

for (int i = 0; i < albero.chiamato.Length; i++)
{

    //
    //occupiamoci degli attributi di Vertex
    //

    if (albero.chiamato[i] == 0) //controllo se attributo è stato
        //chiamato da utente
    {

        comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) ??
VALUES('{0}', '{1}')", "B", albero.getNome());
        //settiamo connessione
        cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
```

```
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
        //query completata
    }

    if (albero.chiamato[i] == 1)
    {
        comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) >
            VALUES('{0}', '{1}')", "C", albero.getNome());
        //settiamo connessione
        cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
        //query completata
    }

    if (albero.chiamato[i] == 2)
    {

        comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) >
            VALUES('{0}', '{1}')", "D", albero.getNome());
        //settiamo connessione
        cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
        //query completata
    }

    if (albero.chiamato[i] == 3)
    {

        comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) >
            VALUES('{0}', '{1}')", "E", albero.getNome());
        //settiamo connessione
        cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
    }

    //
    //occupiamoci degli attributi di Edge
    //
```

```
if (albero.chiamato[i] == 4)
{
    comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) >
    VALUES('{0}', '{1}')", "G", albero.getNome());
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata
}

if (albero.chiamato[i] == 5)
{
    comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) >
    VALUES('{0}', '{1}')", "H", albero.getNome());
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata
}

if (albero.chiamato[i] == 6)
{
    comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) >
    VALUES('{0}', '{1}')", "I", albero.getNome());
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata
}

if (albero.chiamato[i] == 7)
{
    comando = string.Format("INSERT INTO AttrDef(Name, NomeAlbero) >
    VALUES('{0}', '{1}')", "L", albero.getNome());
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
```

```
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
        //query completata
    }
}

//occupiamoci degli AttrDefUid
//

//saranno le chiavi univoche prese dal database
long chiaveAttr1Int = 0;
long chiaveB = 0;
long chiaveC = 0;
long chiaveD = 0;
long chiaveE = 0;
long chiaveAttr2Int = 0;
long chiaveG = 0;
long chiaveH = 0;
long chiaveI = 0;
long chiaveL = 0;

//i due array che tengono il nome dell'attributo e la chiave ad esso    ↵
    associata
string[] Attributi = new string[10];
long[] chiavi = new long[10];

long d = 0; //contatore che tiene la posizione degli array
long letti = 0; //contatore che tiene le linee lette dal database

comando = string.Format("SELECT AttrDefUid, Name FROM AttrDef WHERE    ↵
    NomeAlbero='{0}' ORDER BY Name ASC", albero.getNome());

//settiamo connessione
cmd = new SqlCommand();
cmd.CommandType = System.Data.CommandType.Text;
cmd.CommandText = comando;
cmd.Connection = myconn;

SqlDataReader msdr = cmd.ExecuteReader();
while (msdr.Read())
{
    chiavi[d] = msdr.GetInt32(0);
    Attributi[d] = msdr.GetString(1);
    d++;
    letti++;
}
msdr.Close();

//inserisco le chiavi nelle variabili
for (d = 0; d < letti; d++)
{
```

```
    if (Attributi[d].Equals("attr1Int"))
    {
        chiaveAttr1Int = chiavi[d];
    }

    if (Attributi[d].Equals("B"))
    {
        chiaveB = chiavi[d];
    }

    if (Attributi[d].Equals("C"))
    {
        chiaveC = chiavi[d];
    }

    if (Attributi[d].Equals("D"))
    {
        chiaveD = chiavi[d];
    }

    if (Attributi[d].Equals("E"))
    {
        chiaveE = chiavi[d];
    }

    if (Attributi[d].Equals("attr2Int"))
    {
        chiaveAttr2Int = chiavi[d];
    }

    if (Attributi[d].Equals("G"))
    {
        chiaveG = chiavi[d];
    }

    if (Attributi[d].Equals("H"))
    {
        chiaveH = chiavi[d];
    }

    if (Attributi[d].Equals("I"))
    {
        chiaveI = chiavi[d];
    }

    if (Attributi[d].Equals("L"))
    {
        chiavel = chiavi[d];
    }

}
```

```
//  
//Inseriamo con il for, per farlo in ogni vertice dell' albero!  
//  
  
long IdAlbero = 0; //futuro IdAlbero  
  
long EdgeUid = 0; //variabile che conterrà uno ad uno gli Id Edge ↵  
// presi in considerazione  
long VertexUid = 0; //variabile che conterrà uno ad uno gli Id ↵  
// Vertex presi in considerazione  
  
for (long i = 0; i < albero.albero.Length; i++) //for che scorre ↵  
// tutti i vertici  
{  
  
//  
//occupiamoci dell'Id  
//  
  
comando = string.Format("SELECT Id FROM Albero WHERE  
Nome='{0}'", albero.getNome());  
//settiamo connessione  
cmd = new SqlCommand();  
cmd.CommandType = System.Data.CommandType.Text;  
cmd.CommandText = comando;  
cmd.Connection = myconn;  
  
msdr = cmd.ExecuteReader();  
if (msdr.Read())  
{  
    IdAlbero = msdr.GetInt32(0); //Preleviamo l'IdAlbero  
}  
msdr.Close();  
  
//  
//occupiamoci degli Edge  
//  
  
comando = string.Format("INSERT INTO Edge(Valore, IdAlbero) ↵  
VALUES({0}, {1})", albero.albero[i].arcoentrante.val,  
IdAlbero);  
//settiamo connessione  
cmd = new SqlCommand();  
cmd.CommandType = System.Data.CommandType.Text;  
cmd.CommandText = comando;  
cmd.Connection = myconn;  
  
cmd.ExecuteNonQuery();  
//query di valore e nome_albero si Edge[i] completata
```

```
//  
//occupiamoci dell'EdgeUid  
//  
  
comando = string.Format("SELECT EdgeUid FROM Edge WHERE  
    IdAlbero={0} ORDER BY EdgeUid DESC", IdAlbero);  
//settiamo connessione  
cmd = new SqlCommand();  
cmd.CommandType = System.Data.CommandType.Text;  
cmd.CommandText = comando;  
cmd.Connection = myconn;  
  
msdr = cmd.ExecuteReader();  
if(msdr.Read())  
{  
    EdgeUid = msdr.GetInt32(0); //Preleviamo l'EdgeUid  
}  
msdr.Close();  
  
//  
//occupiamoci dei valori di Vertex  
//  
  
comando = string.Format("INSERT INTO Vertex(Name, Arcoentrante,  
    IdAlbero) VALUES('{0}',{1},'{2}')", albero.albero[i].nome, //  
    EdgeUid, IdAlbero);  
  
//settiamo connessione  
cmd = new SqlCommand();  
cmd.CommandType = System.Data.CommandType.Text;  
cmd.CommandText = comando;  
cmd.Connection = myconn;  
  
cmd.ExecuteNonQuery();  
//query di nome, valore e nome_albero completata su albero[i]  
  
//  
//occupiamoci dei VertexUid  
//  
  
comando = string.Format("SELECT VertexUid FROM Vertex WHERE  
    IdAlbero={0} ORDER BY VertexUid DESC", IdAlbero);  
//settiamo connessione  
cmd = new SqlCommand();  
cmd.CommandType = System.Data.CommandType.Text;  
cmd.CommandText = comando;  
cmd.Connection = myconn;  
  
msdr = cmd.ExecuteReader();
```

```
    if (msdr.Read())
    {
        VertexUid = msdr.GetInt32(0); //Preleviamo il VertexUid
    }

    msdr.Close();

    //
    //Occupiamoci di VertexAttrUsage
    //

    comando = string.Format("INSERT INTO VertexAttrUsage(ObjectVUid,
        AttrDefUid, Value) VALUES({0}, {1}, '{2}')", VertexUid,
        chiaveAttr1Int, albero.albero[i].attr1Int);
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata

    comando = string.Format("INSERT INTO EdgeAttrUsage(ObjectEUid,
        AttrDefUid, Value) VALUES({0},{1},'{2}')", EdgeUid,
        chiaveAttr2Int, albero.albero[i].arcoentrante.attr2Int);
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata

    //for che inserisce nel DB solo gli attributi chiamati dall'utente
    for (int l = 0; l < albero.chiamato.Length; l++)
    {

        if (albero.chiamato[l] == 0)
        {
            comando = string.Format("INSERT INTO VertexAttrUsage
                (ObjectVUid, AttrDefUid, Value) VALUES({0}, {1}, '{2}')",
                VertexUid, chiaveB, albero.albero[i].B);
            //settiamo connessione
            cmd = new SqlCommand();
            cmd.CommandType = System.Data.CommandType.Text;
            cmd.CommandText = comando;
            cmd.Connection = myconn;
```

```
        cmd.ExecuteNonQuery();
        //query completata
    }

    if (albero.chiamato[1] == 1)
    {

        comando = string.Format("INSERT INTO VertexAttrUsage
(ObjectVUid, AttrDefUid, Value) VALUES({0}, {1}, '{2}')",
VertexUid, chiaveC, albero.albero[i].C);
        //settiamo connessione
        cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
        //query completata
    }

    if (albero.chiamato[1] == 2)
    {
        comando = string.Format("INSERT INTO VertexAttrUsage
(ObjectVUid, AttrDefUid, Value) VALUES({0}, {1}, '{2}')",
VertexUid, chiaveD, albero.albero[i].D);
        //settiamo connessione
        cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
        //query completata
    }

    if (albero.chiamato[1] == 3)
    {

        comando = string.Format("INSERT INTO VertexAttrUsage
(ObjectVUid, AttrDefUid, Value) VALUES({0}, {1}, '{2}')",
VertexUid, chiaveE, albero.albero[i].E);
        //settiamo connessione
        cmd = new SqlCommand();
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = comando;
        cmd.Connection = myconn;

        cmd.ExecuteNonQuery();
        //query completata
        //Query VertexAttrUsage completata
    }

//
```

```
//adesso EdgeAttrUsage
//



if (albero.chiamato[1] == 4)
{

    comando = string.Format("INSERT INTO EdgeAttrUsage
(ObjectEUid, AttrDefUid, Value) VALUES({0},{1},'{2}')",
EdgeUid, chiaveG, albero.albero[i].arcoentrante.G);
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata
}

if (albero.chiamato[1] == 5)
{

    comando = string.Format("INSERT INTO EdgeAttrUsage
(ObjectEUid, AttrDefUid, Value) VALUES({0},{1},'{2}')",
EdgeUid, chiaveH, albero.albero[i].arcoentrante.H);
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata
}

if (albero.chiamato[1] == 6)
{

    comando = string.Format("INSERT INTO EdgeAttrUsage
(ObjectEUid, AttrDefUid, Value) VALUES({0},{1},'{2}')",
EdgeUid, chiaveI, albero.albero[i].arcoentrante.I);
    //settiamo connessione
    cmd = new SqlCommand();
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.CommandText = comando;
    cmd.Connection = myconn;

    cmd.ExecuteNonQuery();
    //query completata
}
```

```
        if (albero.chiamato[1] == 7)
        {

            comando = string.Format("INSERT INTO EdgeAttrUsage
( ObjectEUid, AttrDefUid, Value) VALUES({0},{1},'{2}')",
EdgeUid, chiaveL, albero.albero[i].arcoentrante.L);
            //settiamo connessione
            cmd = new SqlCommand();
            cmd.CommandType = System.Data.CommandType.Text;
            cmd.CommandText = comando;
            cmd.Connection = myconn;

            cmd.ExecuteNonQuery();
            //query completata
        }

    } //fine ciclo for array chiamato

} //fine ciclo for array vertici

//importazione in DB è stata completata

myconn.Close(); //chiusura connessione

//Riuscita compito
MessageBox.Show("Update su database completato", "GUI",
MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

}

catch (Exception ex2)
{
    //fallimento compito
    MessageBox.Show("ERRORE: " + ex2.Message);
    MessageBox.Show("Update su database non riuscito", "GUI",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
}

}

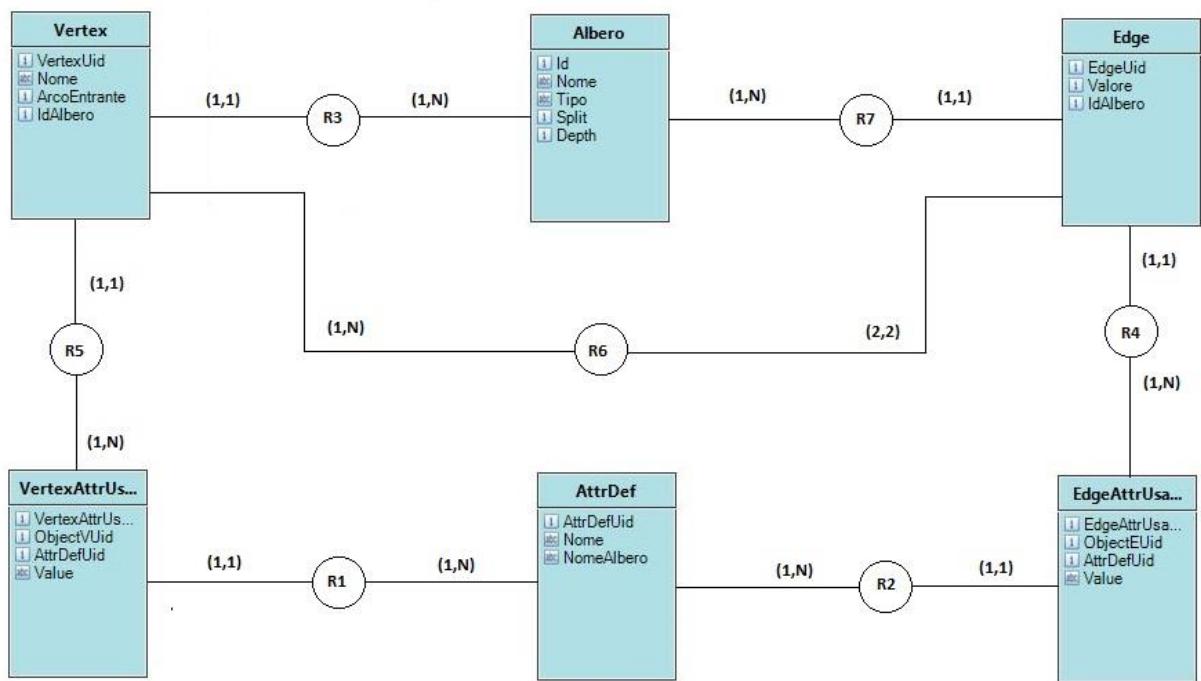
private void pictureBox1_Click(object sender, EventArgs e)
{

}

private void pictureBox2_Click(object sender, EventArgs e)
{

}
}
```

14. E-R Design



15. Code Engine

(Codice Engine completo reperibile su
http://github.com/patriziopezzilli/SQLR/tree/master/SQLR_V4)

L'Engine, fulcro del nostro sistema, nonché obiettivo primario del customer, ha portato il team (e più in particolare il sottoteam che se ne è occupato in modo specifico) a ragionare in modo intensivo in particolare sulla struttura dati da utilizzare, in modo da rendere il tutto efficiente e compatibile con C#, linguaggio sulla quale i membri componenti dei sottogruppi che si occupavano di GUI ed ENGINE si sono dovute documentare in quanto mai affrontato prima.

In un primo approccio si era pensato di poter utilizzare il 'TreeMap' di Java, per rendere le cose più compatibili possibile con un grafo vero e proprio, ma purtroppo tale pista è stata abbandonata in principio poiché nel linguaggio C# non è presente il costrutto 'TreeMap' e implementarcelo ci sarebbe costato troppo tempo in termini di scadenze.

Una seconda ipotesi a cui aveva pensato il sottoteam era quella di implementare una `LinkedList()` di `LinkedList`, o un `ArrayList()` di `ArrayList`, in modo da poter inserire nelle prime i vertici padri e nelle seconde i vertici figli. Anche quest'ipotesi, come la prima, ha avuto vita breve poiché l'`ArrayList` non è implementabile in C# e poi venivano fuori problemi circa l'indicizzazione della `LinkedList` che ci doveva consentire di risalire al padre e viceversa.

Dopo vari ragionamenti e confronti, pensando anche proprio a ciò che abbiamo studiato negli anni precedenti in Algoritmi e Strutture Dati, il team ha deciso di utilizzare una struttura dati semplice come l'`Array` monodimensionale, contenente oggetti `Vertex`.

Tale struttura ci permette di indicizzare alla perfezione e risalire alla cella desiderata con estrema facilità, nonchè accedere a contenuti di padri e figli.

L'`Array` di vertici sarà contenuto nel costruttore della classe 'Tree' e sarà composto da oggetti di tipo `Vertex`.

L'oggetto Vertex sarà costituito da:

- String Nome: che indicherà il nome.
- Int Indice: che sarà fondamentale nell'indicizzazione.
- Double Valore: che useremo come variabile d'appoggio per inserire numeri random tramite funzione C#.
- String[] attributi: che conterrà una lista ordinata o meno di attributi testuali.
- Obj Edge: ovvero il team ha fatto l'assunzione che essendo l'albero N-ARIO ogni vertice avrà solo un arco entrante e quindi per guadagnare in efficienza è stato preferito inserire l'arco entrante nel vertice piuttosto che creare un array a parte che avrebbe contenuto solamente per gli archi.

L'oggetto Edge sarà costituito da:

(che utilizzeremo per trovare il peso dell'arco tramite apposita funzione)

- Int Val: che indica il valore dell'arco.
- String[] attr: che rappresenta gli attributi dell'arco (contenuti in un range come nei Vertex).

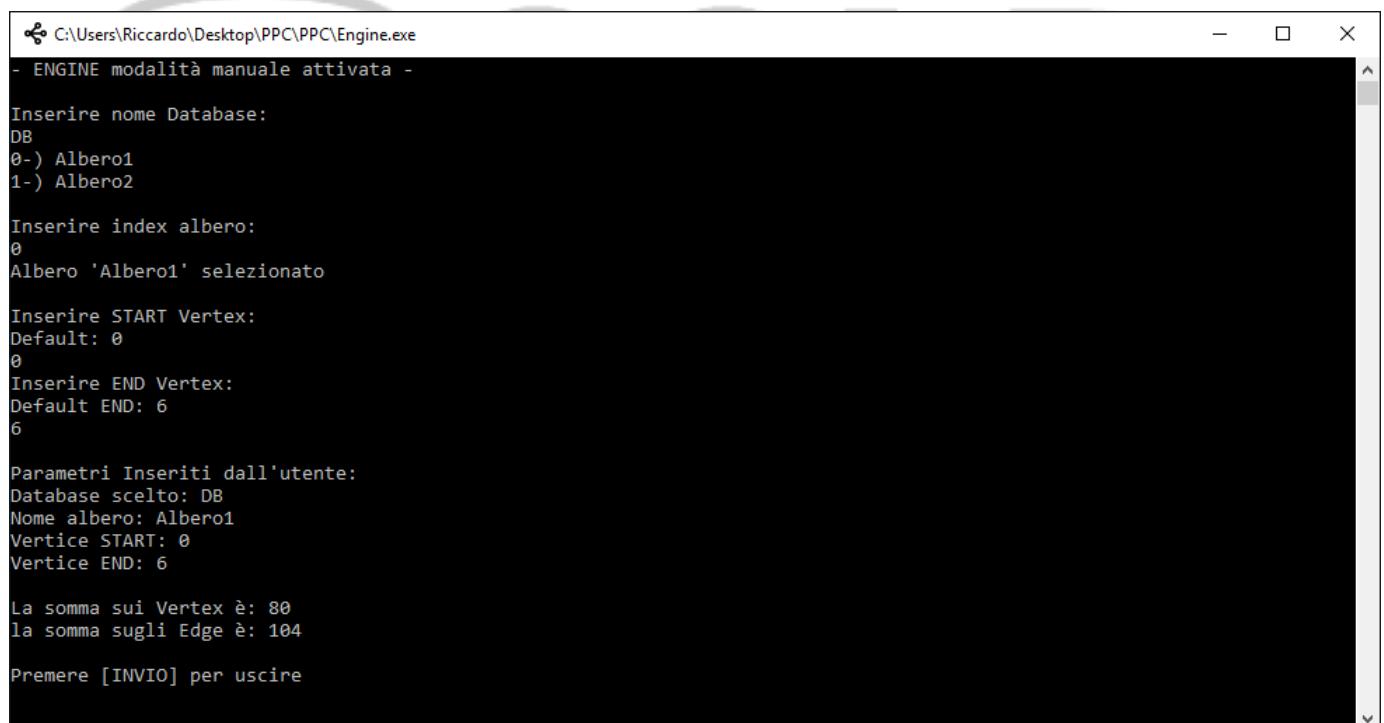
L'oggetto Tree sarà costituito da:

- String Name: che indicherà il nome dell'albero.
- String Type: che indicherà il tipo (contenuti in un range)
- Double Split-size e Depth

Riguardo il calcolo effettivo e l'algoritmo da usare, il team ha effettuato le seguenti considerazioni:
Dati due vertici, si ricerca tramite indice il vertice di arrivo e tramite la formula $(i-1) / \text{depth}$ si risale al padre e confrontando se quest'ultimo corrisponde effettivamente al vertice di partenza eseguiamo il calcolo, altrimenti proseguiamo il cammino verso l'alto.

Una volta uscito dall'array sappiamo che non ci resta che verificare che vertex si quello 'root'.

Engine ESTERNO: La funzione 'Engine', oltre che essere richiamata nella GUI (come richiesto da customer), è richiamabile anche esternamente da altre GUI o tramite linea di comando, effettuando una connessione al DBMS e passandogli i parametri in ingresso via 'Command Line'.

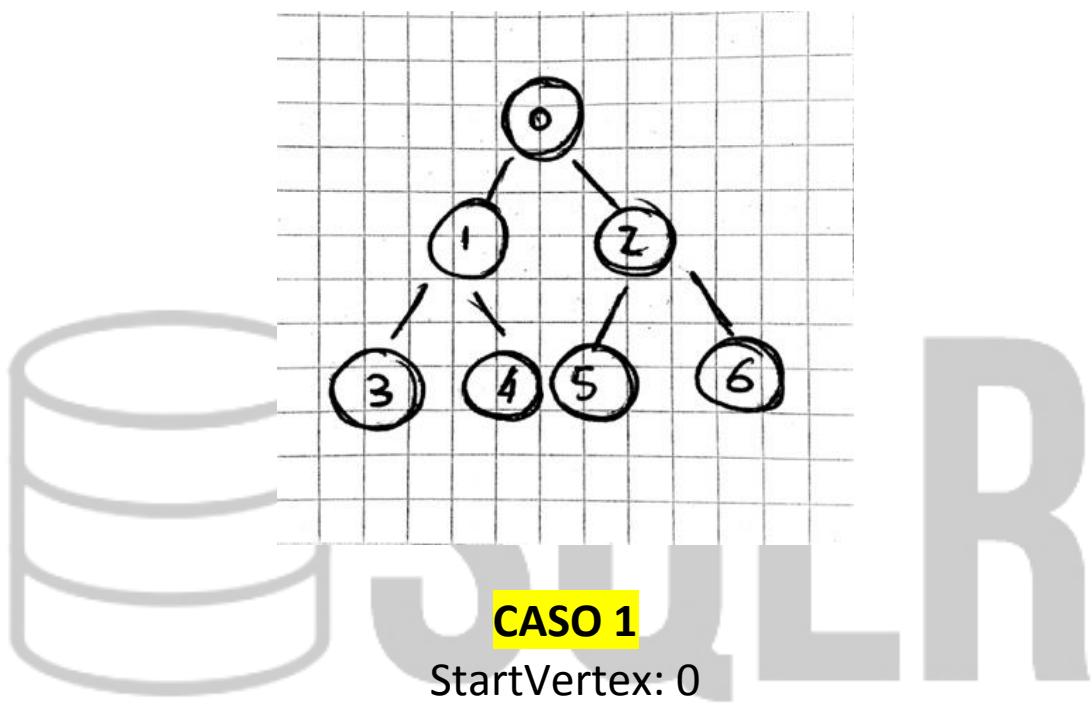


The screenshot shows a terminal window with the following text output:

```
C:\Users\Riccardo\Desktop\PPC\PPC\Engine.exe
- ENGINE modalità manuale attivata -
Inserire nome Database:
DB
0-) Albero1
1-) Albero2
Inserire index albero:
0
Albero 'Albero1' selezionato
Inserire START Vertex:
Default: 0
0
Inserire END Vertex:
Default END: 6
6
Parametri Inseriti dall'utente:
Database scelto: DB
Nome albero: Albero1
Vertice START: 0
Vertice END: 6
La somma sui Vertex è: 80
la somma sugli Edge è: 104
Premere [INVIO] per uscire
```

NOTA BENE: l'algoritmo come definito nelle assunzioni funziona solo nel caso in un cui lo start vertex sia padre o parente dell'end vertex, che dovrà essere a sua volta discendente dello start . Nel caso in cui non lo fosse il programma non risponderebbe.

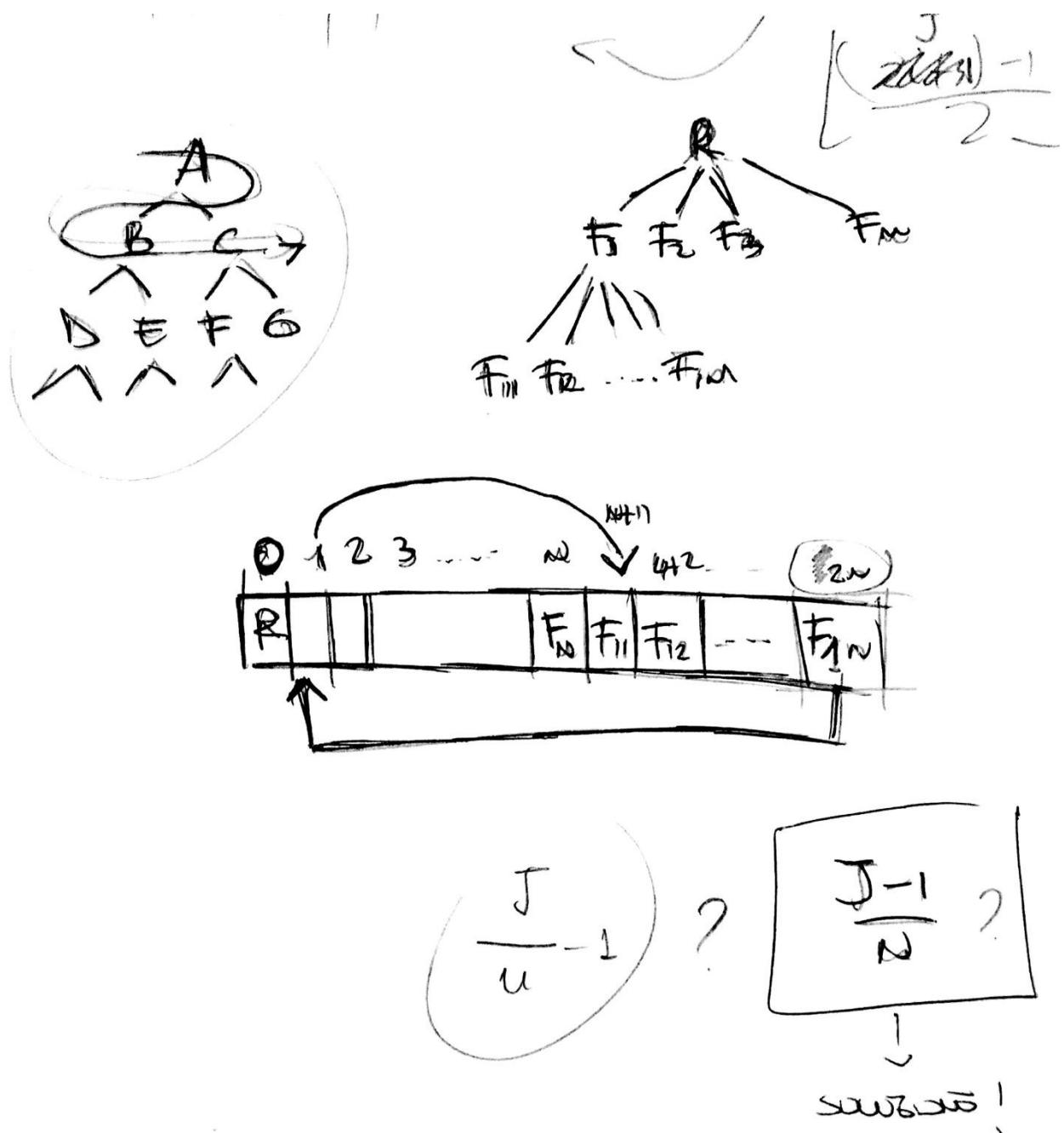
ESEMPIO:

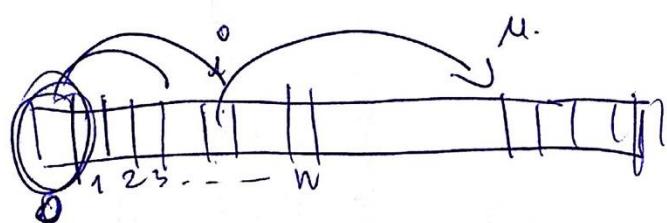
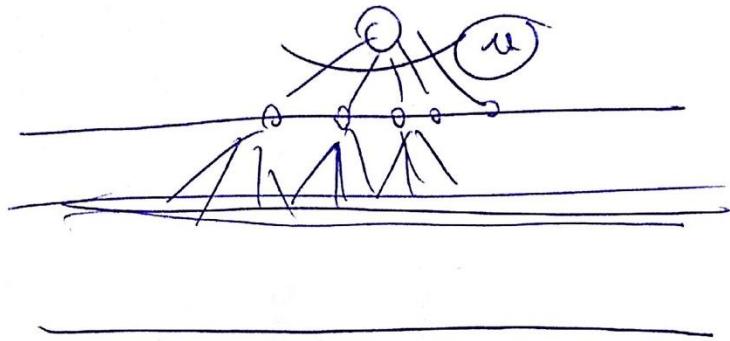


CASO 2
StartVertex: 1
EndVertex: 6
CALCOLO NON RIUSCITO

CASO 3
StartVertex: 4
EndVertex: 0
CALCOLO NON RIUSCITO

Ecco allegati appunti seguiti dal codice effettivo:

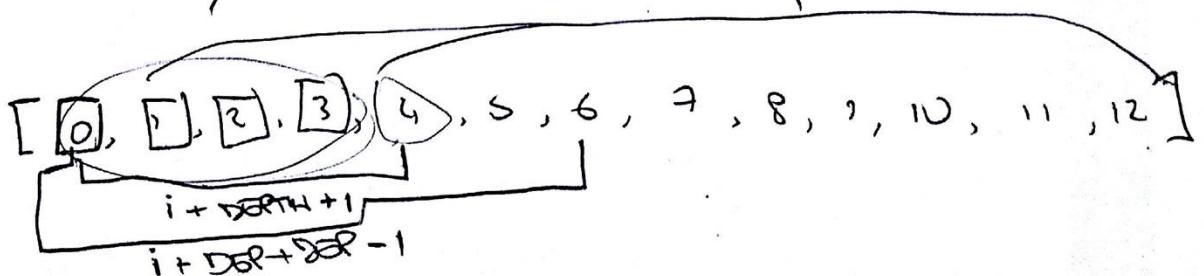
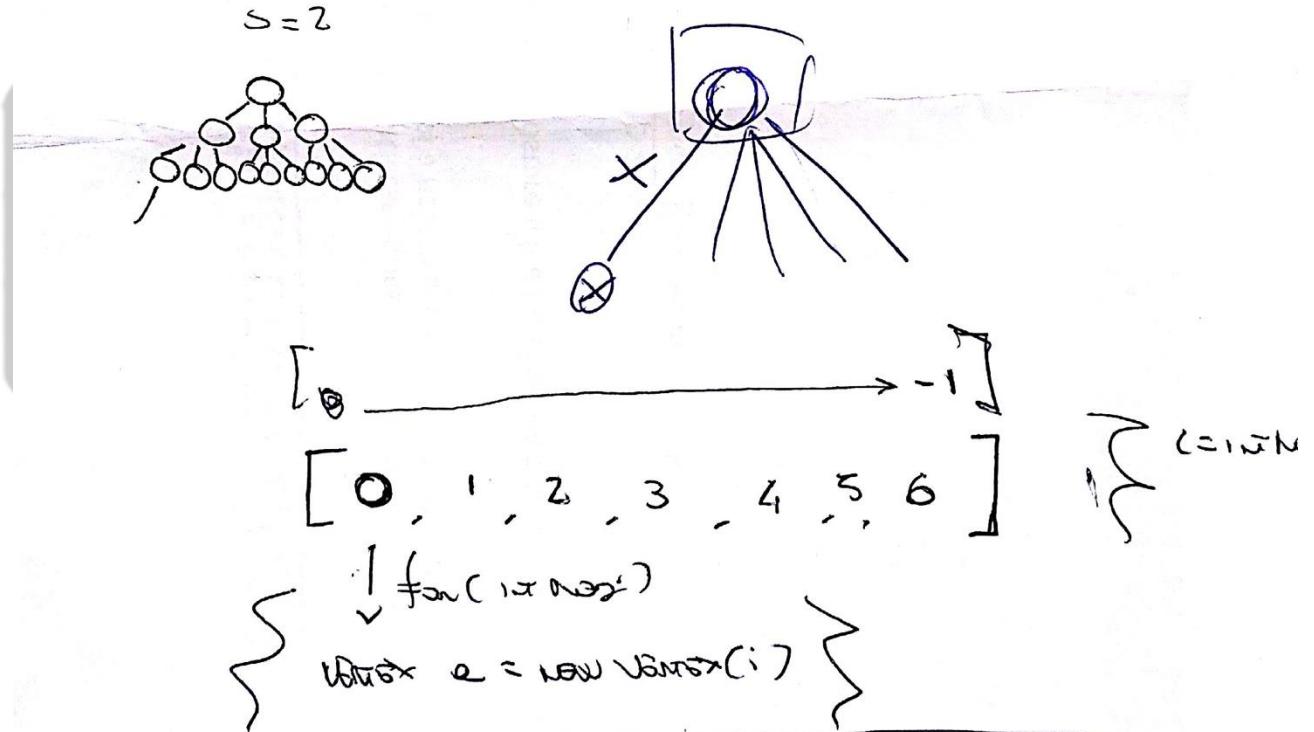




ARRAY DEI PADRI
L'INFO ARCO ATTRIBUITA
AL NODO IN CUI
L'ARCO ENTRA

$$\text{DOP} = 3$$

$$S = 2$$



CODE

Funzione Esecuzione Calcolo (Algoritmo):

```
public long[] EsecuzioneCalcolo(string a, string b)
{
    long contver = 0;                                //contatore vertici
    long contArc = 0;                                //contatore archi
    long i = 1;

    //Errore, contArc non viene ritornato. Possibile soluzione: ritornare array contenente contver e contArc
    long[] result = new long[2];

    if (this.albero[0].getNome() == a)                //caso in cui 'a' è vertice root
    {
        long k = this.getSize();                      //variabile 'k' inizializzata alla grandezza dell'albero
        for (i = 1; this.albero[i].getNome() != b; i++) { } //caso vertice padre
        // ciclo 'for' utilizzato per trovare l'indice di destinazione

        contver += this.albero[i].attr1Int;            //incrementa il risultato con il valore del vertice attuale
        contArc += this.albero[i].arcoentrante.attr2Int; //incrementa il risultato con il peso dell'arco in considerazione

        // siamo usciti dal 'for' quindi la variabile 'i' conterrà l'indice del vertice di destinazione //

        while (this.albero[i].getNome() != a)           //sale l'albero finchè trova il vertice di partenza passatogli
        {
            i = ((i - 1) / (this.depth));              //risale al padre del vertice attuale

            if (i < 0) break;                         //se i < 0 il vertice startVertex è il vertice root

            contver += this.albero[i].attr1Int;
            contArc += this.albero[i].arcoentrante.attr2Int;
        }

        if (i < 0)
        {
            i = 0;

            contver += this.albero[i].attr1Int;
            contArc += this.albero[i].arcoentrante.attr2Int;
        }
        else
        {
            contver += this.albero[i].attr1Int;
            contArc += this.albero[i].arcoentrante.attr2Int;
        }
    }

    result[0] = contArc;
    result[1] = contver;

    return result;
    //return contver;
    //return contArc;
}
else
{
    for (i = 0; this.albero[i].getNome() != b; i++) { } //ciclo per trovare vertice di destinazione

    contver += this.albero[i].attr1Int;                  //incrementa il risultato con il valore del vertice attuale
    contArc += this.albero[i].arcoentrante.attr2Int;     //incrementa il risultato con il peso dell'arco in considerazione

    //siamo usciti dal 'for' quindi la variabile 'i' conterrà l'indice del vertice di destinazione//
    i = ((i - 1) / (this.depth));

    while (this.albero[i].getNome() != a)                //sale l'albero finchè trova il vertice di partenza
    {
        //risale al padre del vertice attuale
        contver += this.albero[i].attr1Int;
        contArc += this.albero[i].arcoentrante.attr2Int;

        i = ((i - 1) / (this.depth));
    }
    contver += this.albero[i].attr1Int;
    contArc += this.albero[i].getPesoArco();

    result[0] = contArc;
    result[1] = contver;

    return result;
    //return contver;
    //return contArc;
}
}
```

Main:

```
static void Main(string[] args)
{
    Tree alberonostro;

    try
    {

        if (args.Length == 0)
        {
            Console.WriteLine("- ENGINE modalità manuale attivata - " + Environment.NewLine);

            Console.WriteLine("Inserire nome Database:");
            string DB = Console.ReadLine();

            //Settaggio connessione a database
            SqlConnection myconn = new SqlConnection(string.Format("Persist Security Info=False;Integrated Security=true;Initial Catalog={0};server=(local)", DB));

            SqlCommand cmd = null; //comando di select
            SqlDataReader myReader = null;

            //Tentativo di connessione al database
            myconn.Open();
            //connessione stabilita

            //numero alberi in DB
            int numeroAlberi = 0;

            cmd = new SqlCommand("SELECT Nome FROM Albero", myconn); //COMANDO SQL

            //settaggio un datareader sul comando appena eseguito
            myReader = cmd.ExecuteReader();

            //Lettura dati dal database
            while (myReader.Read())
            {
                numeroAlberi++;
            }

            // Chiusura DataReader
            myReader.Close();

            //Stampa alberi in DB
            cmd = new SqlCommand("SELECT Nome FROM Albero", myconn); //COMANDO SQL

            //settaggio datareader sul comando appena eseguito
            myReader = cmd.ExecuteReader();

            int[] index = new int[numeroAlberi];
            string[] trees = new string[numeroAlberi];

            int count = 0;
            //Lettura dati dal database
            while (myReader.Read())
            {
                index[count] = count;
                trees[count] = myReader.GetString(0);

                Console.WriteLine((count++) + " - " + myReader.GetString(0));
            }

            //Chiusura DataReader
            myReader.Close();

            Console.WriteLine(Environment.NewLine + "Inserire index albero:");
            int scelta = int.Parse(Console.ReadLine());

            string nalbero = trees[scelta];

            Console.WriteLine("Albero '{0}' selezionato", nalbero);

            int numvertex = 0; //contatore per il numero di vertici

            SqlConnection conn = new SqlConnection(string.Format("Persist Security Info=False;Integrated Security=true;Initial Catalog={0};server=(local)", DB));
            string s = string.Format("SELECT * FROM Albero, Vertex WHERE Albero.Nome='{0}' AND Vertex.IdAlbero=Albero.Id ", nalbero);
            conn.Open();
            SqlCommand msc = new SqlCommand(s, conn);
            SqlDataReader msdr = msc.ExecuteReader();
        }
    }
}
```

```

while (msdr.Read())
{
    numvertex++; //contatore numero vertici
}
msdr.Close();
conn.Close();

Console.WriteLine(Environment.NewLine + "Inserire START Vertex: " + Environment.NewLine + "Default: 0");
int start = Int32.Parse(Console.ReadLine());
Console.WriteLine("Inserire END Vertex: " + Environment.NewLine + "Default END: {0}", (numvertex-1));
int end = Int32.Parse(Console.ReadLine());

//Controllo RANGE
if(!(start >= 0 && start< end && start < numvertex))
{
    //Range startVertex sbagliato!
    Console.WriteLine("Sono stati inseriti parametri non corretti");
    System.Threading.Thread.Sleep(5000);
    return;
}

if (!(end > 0 && end > start && end < numvertex))
{
    //Range endVertex sbagliato!
    Console.WriteLine("Sono stati inseriti parametri non corretti");
    System.Threading.Thread.Sleep(5000);
    return;
}

//verifica andata a buon fine, dati OK

Console.WriteLine(Environment.NewLine + "Parametri Inseriti dall'utente: " + Environment.NewLine + "Database scelto: " +
DB + Environment.NewLine + "Nome albero: " + nalbero + Environment.NewLine + "Vertice START: " + start + Environment.NewLine + "Vertice END: " + end);

//creazione istanza albero tramite riceviDB()
alberonostro = riceviDB(nalbero, DB);

string nstart = alberonostro.albero[start].getNome();
string nEnd = alberonostro.albero[end].getNome();

| long[] res = alberonostro.EsecuzioneCalcolo(nstart, nEnd);

long resvertex = res[1];
long resEdge = res[0];

Console.WriteLine(Environment.NewLine + "La somma sui Vertex è: {0}" + Environment.NewLine + "la somma sugli Edge è: {1}", resVertex, resEdge);

Console.WriteLine(Environment.NewLine + "Premere [INVIO] per uscire");
Console.ReadLine();
return;
}

if (!(args[0].Equals("GUI")))
{
    Console.WriteLine("ERRORE GENERALE: CHIAMATA ENGINE CON PARAMETRO NON ESATTO", args[0]);
    System.Threading.Thread.Sleep(5000);
    return;
}

if (args[0].Equals("GUI")) Console.WriteLine("RILEVATA CHIAMATA DA GUI. Modalità automatica attivata");
//Se chiamato da GUI continuo altrimenti no

alberonostro = new Tree();           //albero sul quale eseguire il calcolo

//Creazione pipe che si aspetta dati dalla GUI
NamedPipeServerStream Pipe1 = new NamedPipeServerStream("Pipe1");

Console.WriteLine("Attendendo connessione con GUI...");
//aspetto che la GUI si connetta alla pipe
Pipe1.WaitForConnection();

Console.WriteLine("Connessione con la GUI stabilita");

//ricezione messaggio
messaggio ricevuto = new messaggio(); //Classe che conterrà Albero, vertice START e vertice END e il futuro risultato
var f = new System.Xml.Serialization.XmlSerializer(typeof(messaggio));
ricevuto = (messaggio)f.Deserialize(Pipe1);

```

```

//chiusura pipe ricezione dalla GUI
Pipe1.Close();

alberonostro = ricevuto.Albero;

string strv = alberonostro.albero[ricevuto.startVertex].getNome();
string endv = alberonostro.albero[ricevuto.endVertex].getNome();

//calcoli su Albero

long[] risultati = alberonostro.EsecuzioneCalcolo(strv, endv);
//l'array risultati conterrà la somma su attrInt e attr2Int, prendendo il relativo nome (con la funzione getName()) dalla posizione nell'array di vertex identificato dall'indice (startVertex o endvertex)

string msg = string.Format("La somma sui Vertici è: {0}" + Environment.NewLine + "La somma sugli Edge è: {1}", risultati[1], risultati[0]); //risultato (Conteggio su Vertici, conteggio su Archi) da mandare

//invio messaggio tramite pipe
NamedPipeClientStream Pipe2 = new NamedPipeClientStream("Pipe2"); //creazione pipe che invia dati alla GUI
Thread.Sleep(1000);
Pipe2.Connect();

//creazione classe messaggio che conterrà il risultato da mandare
messaggio invio = new messaggio();
invio.Risposta = msg;
f.Serialize(Pipe2, invio); //invio dati alla GUI

//chiusura pipe invio e uscita
Pipe2.Close();

}

catch (Exception ex)
{
    //Eccezioni

    if (args.Length == 0) //stampa eccezione a video
    {
        Console.WriteLine("ERROR: " + ex.Message);
        System.Threading.Thread.Sleep(5000);
        return;
    }

    //se Engine chiamato dalla GUI invia l'eccezione ad essa
    var f = new System.Xml.Serialization.XmlSerializer(typeof(messaggio));

    string msg = "ERROR IN ENGINE: " + ex.Message;
    //invio messaggio errore tramite pipe
    NamedPipeClientStream Pipe2 = new NamedPipeClientStream("Pipe2"); //creazione pipe che invia dati alla GUI
    Thread.Sleep(1000);
    Pipe2.Connect();

    //creazione casse messaggio che conterrà il risultato da mandare
    messaggio invio = new messaggio();
    invio.Risposta = msg;
    f.Serialize(Pipe2, invio); //invio i dati alla GUI

    //chiusura pipe invio e uscita
    Pipe2.Close();
}
}

```

16.Code DBMS

(Codice DBMS reperibile su file DUMP.txt)

```
create table Albero
(
Id int identity,
Nome varchar(50) unique,
Tipo varchar(50),
Split int,
Depth int,
CONSTRAINT [Albero_pk] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)
)
```

```
create table Edge
(
EdgeUid int identity,
Valore int,
IdAlbero int,
CONSTRAINT [Edge_pk] PRIMARY KEY CLUSTERED
(
    [EdgeUid] ASC
)
)
```

```
ALTER TABLE Edge WITH CHECK ADD CONSTRAINT [edge_to_albero] FOREIGN
KEY([IdAlbero])
REFERENCES [Albero] ([Id])
```

```
create table Vertex
(
VertexUid int identity,
Name varchar(100),
```

```
ArcoEntrante int,  
IdAlbero int,  
CONSTRAINT [Vertex_pk] PRIMARY KEY CLUSTERED  
(  
    [VertexUid] ASC  
)  
)
```

```
ALTER TABLE Vertex WITH CHECK ADD CONSTRAINT [vertex_to_albero] FOREIGN  
KEY([IdAlbero])  
REFERENCES [Albero] ([Id])
```

```
ALTER TABLE Vertex WITH CHECK ADD CONSTRAINT [vertex_to_edge] FOREIGN  
KEY([ArcoEntrante])  
REFERENCES [Edge] ([EdgeUid])
```

```
create table AttrDef  
(  
    AttrDefUid int identity,  
    Name varchar(50),  
    NomeAlbero varchar(50),  
  
CONSTRAINT [AttrDef_pk] PRIMARY KEY CLUSTERED  
(  
    [AttrDefUid] ASC  
)  
)
```

```
ALTER TABLE AttrDef WITH CHECK ADD CONSTRAINT [Albero_to_AttrDef] FOREIGN  
KEY([NomeAlbero])  
REFERENCES [Albero] ([Nome])
```

```
create table VertexAttrUsage  
(  
    VertexAttrUsageId int identity,  
    ObjectVUid int,  
    AttrDefUid int,  
    Value varchar(1000),  
CONSTRAINT [VertexAttrUsage_pk] PRIMARY KEY CLUSTERED  
(  
    [VertexAttrUsageId] ASC  
)
```

```
)
```

```
ALTER TABLE VertexAttrUsage WITH CHECK ADD CONSTRAINT [vertex_attr_usage_to
vertex] FOREIGN KEY([ObjectVUid])
REFERENCES [Vertex] ([VertexUid])
```

```
create table EdgeAttrUsage
(
    EdgeAttrUsageUid int identity,
    ObjectEUid int,
    AttrDefUid int,
    Value varchar(1000),
CONSTRAINT [EdgeAttrUsage_pk] PRIMARY KEY CLUSTERED
(
    [EdgeAttrUsageUid] ASC
)
)
```

```
ALTER TABLE EdgeAttrUsage WITH CHECK ADD CONSTRAINT [edge_attr_usage_to_edge]
FOREIGN KEY([ObjectEUid])
REFERENCES [Edge] ([EdgeUid])
```

17. About Us

Il team SQLR si è formato immediatamente dopo la consegna della specifica del progetto, con i componenti che si sono uniti in base ad una amicizia preesistente. Inizialmente si è svolto un meeting in cui sono stati scelti i giorni in cui lavorare in base agli impegni esterni e quali strumenti utilizzare per condividere documenti e svolgere il lavoro.

Si è scelto di utilizzare la piattaforma “Google Drive” poiché cloud, il quale permette la modifica e l’upload di file o documenti real-time. Per i primi due deliverable la frequenza degli incontri è stata di 1, 2 volte a settimana, nelle quali tutti i membri del team si incontravano per portare avanti il progetto in comune. Dal Deliverable 3 è nata la necessità di creare 3 sottogruppi autonomi ognuno con un compito specifico: lo sviluppo di DBMS, GUI ed ENGINE. Questa fase del progetto è coincisa con il periodo natalizio, che per impossibilità di vederci fisicamente ci ha indotto a lavorare tramite piattaforme di comunicazione online come Skype. Una volta finita la pausa natalizia, prima della consegna della Deliverable 3, i sottogruppi si sono incontrati nuovamente per cooperare al fine di assemblare le sotto componenti sviluppate per rendere il sistema completo ed efficiente.

Infine per l’ultima consegna, quella del deliverable 4, è stato eseguito un raffinamento del lavoro svolto nei tre mesi precedenti con l’aggiunta di specifiche richieste del customer al fine di rendere il sistema corretto e stabile dal punto di vista strutturale.

Questo corso ci ha portato a vedere il software da un diverso punto di vista, più strutturale e meno grafico comprensivo di nuove metodologie di lavoro in particolare sull’organizzazione in team, mai affrontata prima.



E’ stata una bella esperienza ...