SPECIFICHE ELABORATO	
Titolo	**Lettore musicale: riproduzione di brani musicali in base al genere scelto dall'utente**
Descrizione dettagliata	Il progetto riguarda la creazione di una media player, non con l'effettivo download delle canzoni ma semplicemente attraverso i link relativi ai video delle canzoni su YouTube. Questo permette non solo di salvare la memoria di archiviazione, bensì di avere la possibilità di ottenere diverse varianti per una stessa canzone. Questo, anche grazie al fatto che tutto il progetto si basa sull'utilizzo della lista posizionale, che permette di andare a modificare il link relativo alla posizione del vecchio link, per avere tutto correttamente funzionante. Il lettore musicale permette, attraverso le scelte compiute dall'utente, la creazione di una playlist nell'immediato, in modo da poter scegliere i brani che si vogliono ascoltare al momento, dando anche la possibilità di ricercare le canzoni in base al genere musicale. Permette, inoltre, di ascoltare le canzoni più in voga del momento. Come ultimo punto, rende possibile la riproduzione casuale dei brani, in base alla scelta dell'utente. Il programma funziona nella seguente maniera: ● Vengono effettuate delle domande all'utente, in cui viene richiesto, nel seguente ordine: 1. Se si intende ascoltare della musica; 2. In caso di risposta positiva alla domanda posta al punto 1, chiederà se si vogliono ascoltare delle specifiche canzoni, altrimenti, augurerà una buona giornata terminando il programma; 3. In caso di risposta positiva alla domanda posta al punto 2, chiederà se si vuole scegliere un genere a lui/lei gradito; 4. In caso di risposta positiva alla domanda posta al punto 3, farà partire un'interfaccia grafica che permetterà la scelta del genere che si vuole ascoltare. Successivamente, in base al genere scelto, verranno prelevati i brani da una Tabella Hash, e partirà così la riproduzione di questi; 5. In caso di risposta negativa alla domanda posta al punto 2, chiederà se si vogliono ascoltare le canzoni presenti nella top 100;

	6. In caso di risposta affermativa alla domanda posta al punto 5, il programma preleverà dei dati da youtube dando ad ogni canzone una priorità minore, che sarà creata in maniera randomica tramite i dati, alla canzone più ascoltata. Successivamente, le priorità associate ai link verranno inserite all'interno di una struttura dati Heap, la quale ne permetterà l'organizzazione in base alla priorità minore. Partirà così la riproduzione musicale; 7. In caso di risposta negativa alla domanda posta al punto 5, chiederà se si vuole attivare la riproduzione casuale; 8. In caso di risposta negativa alla domanda posta al punto 3, chiederà se si vogliono scegliere manualmente le canzoni; 9. In caso di risposta positiva alla domanda posta al punto 7, chiederà se si gradisce far partire una riproduzione casuale dei brani dando la precedenza alle canzoni più rilassanti; 10. In caso di risposta negativa alla domanda posta al punto 7, chiederà se si gradisce far partire la riproduzione casuale contenente le canzoni disposte senza criteri prestabiliti dall'utente; 11. In caso di risposta negativa alla domanda posta al punto 9, chiederà nuovamente se si gradisce far partire la riproduzione casuale dando la priorità alle canzoni più movimentate; 12. In caso di risposta positiva alle domande riguardanti la riproduzione casuale, verranno scelte randomicamente delle canzoni presenti nella lista posizionale. Successivamente, si andrà a controllare il genere e, in base a questo, si assegnerà una priorità maggiore o minore in base alle scelte dell'utente; 13. In caso di risposta positiva alla domanda posta al punto 8, farà partire un'interfaccia grafica che permetterà la selezione delle canzoni che si vogliono ascoltare e, successivamente, dopo la conferma dell'utente farà partire la riproduzione musicale;

	14. In caso di risposta negativa alla domanda posta al punto 8, augurerà buona giornata. • Vengono riprodotte le playlist in base ai risultati ottenuti dalle domande effettuate all'utente, azionando le seguenti strutture: a. Coda; b. Lista posizionale; c. Albero; d. Coda prioritaria; e. Heap; f. Mappa; g. Tabella Hash.
Strutture dati utilizzate	❏ Coda: utilizzata per la gestione della riproduzione manuale del brani, ossia il passaggio in avanti o indietro dei brani; ❏ Lista posizionale: utilizzata per salvare i link in modo tale che siano facilmente selezionabili conoscendo la loro posizione. Questo permette, in caso di modifica del link ad una determinata posizione, che il link venga aggiornato in tutti i punti in cui viene richiamata la canzone; ❏ Albero: utilizzato come un decision tree, permette all'utente di scegliere, selezionando "sì" oppure "no", quale operazione effettuare; ❏ Coda prioritaria: utilizzata per la gestione dei brani in riproduzione casuale, per cui in base ai gusti dell'utente e ai dati prelevati da youtube, ad ogni canzone prelevata randomicamente dalla lista verrà assegnata una priorità; ❏ Heap: utilizzato per la creazione di una classifica Top Chart, in modo tale che ad ogni canzone venga associato una priorità in base ai dati prelevati da youtube, così questi ultimi vengono ordinati in base alla statistica; ❏ Mappa: la prima utilizzata per associare ad ogni link il relativo nome della canzone e quindi per la gestione e creazione della playlist da parte dell'utente. La seconda viene invece utilizzata per l'associazione link e relativo genere musicale utilizzata successivamente nella coda prioritaria per gestire la priorità in base al gusto dell'utente; ❏ Tabella hash: utilizzata per la gestione delle canzoni

	in base al genere. Difatti ad ogni canzone viene associato un genere che attraverso una funzione di modifica del nome di questo, verrà poi inserito nel bucket corrispondente.
Metodi implementati	**File "decision_tree" (Main) :** **1. Classe "DecisionTree":** ● **loading()**: Metodo pubblico. Permette il caricamento di stringhe all'interno dell'albero. ○ **Parametri di ingresso**: ❖ **decision_tree**. Albero di decisioni costituito dalle domande poste all'utente e dalle sue relative risposte. ○ **Parametri di uscita**: nessuno. ● **tree()**: Metodo pubblico. Permette di far partire l'interfaccia grafica con la quale l'utente può interagire per selezionare le varie scelte rispondendo con la pressione di 2 pulsanti. ○ **Parametri di ingresso**: ❖ **decision_tree**. Albero di decisioni costituito dalle domande poste all'utente e dalle sue relative risposte. ○ **Parametri di uscita**: nessuno. ● **right()**: Metodo pubblico. Permette di scorrere l'albero a destra della posizione in cui si trova selezionando quindi il figlio destro. ○ **Parametri di ingresso**: nessuno. ○ **Parametri di uscita**: nessuno. ● **left()**: Metodo pubblico. Permette di scorrere l'albero a sinistra della posizione in cui si trova selezionando quindi il figlio sinistro. ○ **Parametri di ingresso**: nessuno. ○ **Parametri di uscita**: nessuno. ● **start()**: Metodo pubblico. Permette l'inizializzazione dell'albero e successivamente farà partire l'interfaccia grafica. ○ **Parametri di ingresso**: nessuno. ○ **Parametri di uscita**: nessuno. **2. Classe "ReTree":** ● **restart()**: Metodo pubblico. Permette di far ripartire l'albero alla conclusione della riproduzione dei brani, in modo da permettere all'utente di effettuare altre operazioni.

- **Parametri di ingresso**: nessuno.
- **Parametri di uscita**: nessuno.

File “center”:

Classe “center”:

- **center():** metodo pubblico. Non interagisce con le strutture dati. Permette di andare a posizionare centralmente le interfacce grafiche.
  - **Parametri di ingresso**:
    - **win.** oggetto dell’interfaccia grafica che si vuole posizionare centralmente
  - **Parametri di uscita**: nessuno.

File “file_reader”:

- **read_songs()**: Metodo pubblico. Non interagisce con le strutture dati.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**:
    - **songs_list**. Lista di canzoni presenti nel file “songs.txt”.

- **read_singers()**: Metodo pubblico. Non interagisce con le strutture dati.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**:
    - **singers_list**. Lista di cantanti presenti nel file “songs.txt”.

- **read_genres()**: Metodo pubblico. Non interagisce con le strutture dati.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**:
    - **genres_list**. Lista di generi musicali presenti nel file “songs.txt”.

- **read_modified_genres()**: Metodo pubblico. Non interagisce con le strutture dati.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**:
    - **modified_genres_list**. Lista di generi musicali tratti dal file “songs.txt” e modificati opportunamente nel nome.

- **read_link()**: Metodo pubblico. Interagisce con la lista posizionale.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**: **nessuno;**

- **read_links()**: Metodo pubblico. Permettere di ritornare sempre la stessa lista posizionale
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**:
    - **links_list**: lista posizionale

File "linked_deque":

Classe "LinkedDeque":

- **first2()**:
  - **Parametri di ingresso**:
    - **self.**
  - **Parametri di uscita**:
    - **posizione al primo elemento della coda doppia;**
- **after()**:
  - **Parametri di ingresso**:
    - **p: posizione**
  - **Parametri di uscita**:
    - **elemento dopo la posizione p.**
- **before()**:
  - **Parametri di ingresso**:
    - **p: posizione**
  - **Parametri di uscita**:
    - **elemento prima della posizione p.**

- **position_element()**:
  - **Parametri di ingresso**:
    - **p: posizione**
  - **Parametri di uscita**:
    - **elemento alla posizione p.**

File "positional_list":

Classe "PositionalList":

- **iter2()**: Permette di ritornare un iteratore delle posizioni della lista posizionale
  - **Parametri di ingresso**:
    - **self.**
  - **Parametri di uscita**:
    - **iteratore.**

File "reproduction":

- **play()**: Metodo pubblico. Permette di far partire l'interfaccia grafica che permette di interagire con il player vlc.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**: nessuno

- **backward()**: Metodo pubblico. Permette nel momento in cui viene premuto il pulsante indietro di andare a selezionare la canzone

	precedente. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: nessuno. ● **forward()**: Metodo pubblico. Permette nel momento in cui viene premuto il pulsante avanti di andare a selezionare la canzone successiva. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: nessuno. ● **pause()**: Metodo pubblico. Mette in pausa il player oppure lo fa ripartire. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: nessuno. ● **back_to_tree()**: Metodo pubblico. Permette di chiudere sia il player che l'interfaccia grafica. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: nessuno. ● **return_tree()**: Metodo pubblico. Permette di andar a richiamare il decision tree permettendo all'utente di selezionare altre operazioni. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: nessuno. ● **control()**: Metodo pubblico. Permette di controllare se la coda è finita. ○ **Parametri di ingresso**: ❖ **element**. Elemento che tiene conto del numero della canzone all'interno della playlist. ○ **Parametri di uscita**: nessuno. ● **choices()**: Metodo pubblico. Interagisce con l'heap, la coda, la lista posizionale e la coda prioritaria. Permette di andare a controllare quale delle riproduzioni, e quindi quali delle strutture dati, si sta utilizzando per andare successivamente ad organizzare le canzoni all'interno della coda doppiamente concatenata. ○ **Parametri di ingresso**: ❖ **choice**. Scelta della struttura da utilizzare, tra heap, coda, lista posizionale e coda prioritaria; ❖ **data**. Dati da inserire in una delle strutture sopra elencate.

- **Parametri di uscita**: nessuno.

- **vlc_reproduction()**: Metodo pubblico. Non interagisce con le strutture dati. Permette di inizializzare il player vlc che permette di andare a riprodurre il brano attraverso il link di youtube.
  - **Parametri di ingresso**:
    - **url**. Indirizzo web del video di YouTube associato alla canzone eseguita dal riproduttore.
  - **Parametri di uscita**:
    - **str(title)**. Titolo della canzone, sotto forma di stringa.

- **waiting()**: Metodo pubblico. Interagisce con la coda. Metodo richiamato al verificarsi di un evento del player, esso fa sì che venga selezionato automaticamente il brano successivo.
  - **Parametri di ingresso**:
    - **event**. Evento legato al riproduttore audio.
  - **Parametri di uscita**: nessuno.

- **setting_pause()**: Metodo pubblico. Non interagisce con le strutture dati.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**: nessuno.

- **blocking_function()**: metodo pubblico. Non interagisce con le strutture dati, permette di rimanere in attesa della conclusione della canzone attraverso una flag in modo tale che alla terminazione di essa farà partire il brano successivo.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**: nessuno.

- **start_new_thread()**: metodo pubblico. Non interagisce con le strutture dati. permette di inizializzare e far partire un thread.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**: nessuno.

File “request_data”:

1. Classe “RequestData”:
   - **get_data()**: Metodo pubblico. Non interagisce con le strutture dati. Permette attraverso le API di google di prelevare i dati relativo ai link

	passati come parametro. ○ **Parametri di ingresso**: ❖ **self**. Istanza corrente della classe; ❖ **video_id**. ID del video di YouTube associato alla canzone riprodotta ricavabile dal relativo URL ottenuto attraverso le API di Google; ❖ **part**. Parte di dati del video di YouTube associato alla canzone riprodotta ricavabile dal relativo URL ottenuto attraverso le API di Google; ❖ **api_key**. Chiave del video di YouTube associato alla canzone riprodotta ricavabile dal relativo URL ottenuto attraverso le API di Google. ○ **Parametri di uscita**: ❖ **data**. Insieme dei dati del video di YouTube associato alla canzone riprodotta ricavabile dal relativo URL ottenuto attraverso le API di Google. ● **search()**: Metodo pubblico. Non interagisce con le strutture dati. Permette di andare ad estrapolare dal link di youtube il codice relativo al video stesso. ○ **Parametri di ingresso**: ❖ **self**. Istanza corrente della classe; ❖ **string**. Link del video di YouTube associato alla canzone riprodotta. ○ **Parametri di uscita**: ❖ **substring**. Codice del link del video di YouTube associato alla canzone riprodotta. ● **research()**: Metodo pubblico. Non interagisce con le strutture dati. Permette di andare a richiamare i vari metodi di estrapolazione dei dati. ○ **Parametri di ingresso**: ❖ **self**. Istanza corrente della classe; ❖ **string**. Link del video di YouTube associato alla canzone riprodotta. ○ **Parametri di uscita**: ❖ **data**. Insieme dei dati del link del video di YouTube associato alla canzone riprodotta.

	File “**shuffle_playback**”: ● **case()**: Metodo pubblico. Interagisce con la mappa e la lista posizionale. Permette di associare ad ogni canzone, scelta randomicamente dalla lista posizionale, una priorità in base alla scelta dell’utente. ○ **Parametri di ingresso**: ❖ **choice**. Genere musicale scelto dall’utente. ○ **Parametri di uscita**: nessuno. File “**statistics**”: 1. Classe “**Data**”: ● **__init__()**. File “**structures_filler**”: ● **random_draw()**: Metodo pubblico. Non interagisce con le strutture dati. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: ❖ **random_songs_list**. Lista di canzoni presenti nel file “songs.txt” e disposte casualmente. ● **find_songs()**: Metodo pubblico. Interagisce con la tabella hash. ○ **Parametri di ingresso**: ❖ **chosen_genre**. Genere musicale scelto dall’utente attraverso l’heap. ○ **Parametri di uscita**: ❖ **founded_songs_list**. Lista con tutte le canzoni del genere musicale scelto dall’utente. ● **hash_map_filling()**: Metodo pubblico. Interagisce con la tabella hash. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: ❖ **hash_map**. Tabella hash con tuple composte da coppie (chiave, valore) di tipo (generi musicali modificati, posizioni in memoria dei link). ● **heap_filling()**: Metodo pubblico. Interagisce con l’heap. ○ **Parametri di ingresso**: nessuno; ○ **Parametri di uscita**: ❖ **heap**. Heap con tuple composte da coppie (chiave, valore) di tipo (canzone, cantante).

- **sp_map_filling()**: Metodo pubblico. Interagisce con la mappa.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**:
    - **sp_map**. Mappa con tuple composte da coppie (chiave, valore) di tipo (canzone, posizioni in memoria dei link).

- **pg_map_filling()**: Metodo pubblico. Interagisce con la mappa.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**:
    - **pg_map**. Mappa con tuple composte da coppie (chiave, valore) di tipo (posizioni in memoria dei link, generi musicali).

- **priority_queue_filling()**: Metodo pubblico. Interagisce con la coda prioritaria.
  - **Parametri di ingresso**: nessuno.
  - **Parametri di uscita**:
    - **priority_queue**. Coda prioritaria con tuple composte da coppie (chiave, valore) di tipo (priorità casuale, canzone).

File "tk_genres":

- **menu()**: Metodo pubblico. Non interagisce con le strutture dati. Permette di far partire l'interfaccia grafica che permette all'utente di selezionare i generi.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**: nessuno.

- **selected_item()**: Metodo pubblico. Non interagisce con le strutture dati. Metodo pubblico. Interagisce con la coda. Metodo richiamato al momento in cui viene premuto il pulsante done e permette di andare a prelevare le posizioni delle canzoni all'interno della tabella hash.
  - **Parametri di ingresso**: nessuno;
  - **Parametri di uscita**: nessuno.

File "tk_queue":

- **menu()**: Metodo pubblico. Interagisce con la mappa. Permette di far partire l'interfaccia grafica che permette all'utente di selezionare i

brani.
    - **Parametri di ingresso**: nessuno;
    - **Parametri di uscita**: nessuno.

- **selected_item()**: Metodo pubblico. Interagisce con la coda. Metodo richiamato al momento in cui viene premuto il pulsante done e permette di andare a prelevare le posizioni all'interno della mappa dei vari brani.
    - **Parametri di ingresso**: nessuno;
    - **Parametri di uscita**: nessuno.

File "top_100":

Classe "Top100":

- **classification()**: Metodo pubblico. Non interagisce con le strutture dati. Permette di assegnare una priorità ad ogni canzone.
    - **Parametri di ingresso**:
        - ❖ **data**. Insieme dei dati del link del video di YouTube associato alla canzone riprodotta.
    - **Parametri di uscita**:
        - ❖ **priority**. Valore di priorità assegnato alle canzoni in base al numero di like e commenti delle stesse.

- **start()**: Metodo pubblico. Non interagisce con le strutture dati. Permette di mandare in esecuzione il thread e l'interfaccia grafica.
    - **Parametri di ingresso**: nessuno
    - **Parametri di uscita**: nessuno.

- **load()**: Metodo pubblico. Interagisce con la lista posizionale e l'heap. Permette per ogni brano all'interno della lista posizionale di andare a prelevare i dati da youtube e richiamare la funzione che ne assegna la priorità, modificando anche il testo dell'interfaccia grafica.
    - **Parametri di ingresso**: nessuno
    - **Parametri di uscita**: nessuno.

- **tk()**: Metodo pubblico. Non interagisce con le strutture dati. interfaccia grafica che comunica all'utente di attendere la conclusione dell'acquisizione dei dati.
    - **Parametri di ingresso**: nessuno
    - **Parametri di uscita**: nessuno

	• **start_new_thread()**: Metodo pubblico. Non interagisce con le strutture dati. Permette di inizializzare e mandare in esecuzione un thread.
	  ○ **Parametri di ingresso**: nessuno
	  ○ **Parametri di uscita**: nessuno.