ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

# AlGraph – Algoritmo di Johnson

## Workgroup

Martina Sosto, Andrea Vicenzi, Sara Vorabbi

## Obiettivo primario

Realizzare un software utile che utilizzi l'algoritmo di Johnson per il cammino minimo sui grafi pesati

## Obiettivi secondari

• creare campi/bottoni per inserire dati e interagire con l'applicazione

• permettere di generare un grafo casualmente (dato in input il numero dei nodi), importare da un file esterno e di modificare il grafo corrente

• esportare il grafo e il percorso trovato dall'algoritmo

• permettere di eseguire l'algoritmo step-by-step o visualizzare direttamente il risultato

• gestire correttamente gli errori e conseguente notifica all'utente (input scorretto, errori di selezione, etc…)

• creare un'applicazione dalla grafica piacevole

## Svolgimento del programma

All'inizio del programma l'utente può decidere se generare il grafo casualmente, crearlo da zero oppure importarlo da un file formato .txt esterno. Nel caso in cui il grafo venga creato casualmente, sarà richiesto all'utente di inserire il numero di nodi che andranno a comporre il grafo, con un minimo di 1 nodo fino a 10 nodi massimi.

Una volta creato il grafo, lo si potrà visualizzare in una nuova finestra, nella quale, oltre al disegno del grafo, saranno visibili i collegamenti. Si avranno inoltre a disposizione tre bottoni: uno per l'esportazione del grafo, per la sua modifica e infine, per l'avvio dell'algoritmo.

Nel menù di modifica del grafo l'utente può decidere di:

1. Aggiungere un nodo
2. Eliminare un nodo
3. Aggiungere un arco
4. Eliminare un arco

5. Modificare il peso di un arco
6. Modificare la direzione di un arco

Una volta confermate le modifiche, se queste risultano corrette, sono subito visibili, con i nodi e gli archi aggiornati.

Se premuto il pulsante "Cammino Minimo", verrà avviato l'algoritmo in una nuova schermata. Questo ha due modalità di esecuzione: cliccando il bottone "Avanti" si visualizzerà l'algoritmo step-by-step, grazie al quale l'utente potrà avanzare di un passo nell'esecuzione; cliccando "Mostra direttamente il risultato" si visualizzerà il grafo finale con il cammino minimo trovato. Una volta finito l'algoritmo, l'utente può decidere di esportare il cammino minimo su un file .txt oppure di concludere il programma.

## Scelte implementative

Grazie all'utilizzo di Java, è stato possibile utilizzare la programmazione a oggetti. Abbiamo scelto di implementare il grafo attraverso una HashMap. La HashMap è una classe che possiede due tipi di parametri, la chiave (Key) e i suoi valori (Values) corrispondenti. Al parametro chiave abbiamo deciso di assegnare la classe Node, una classe implementata per gestire i nodi del grafo durante la sua costruzione e durante lo svolgimento dell'algoritmo. Ad ogni nodo viene assegnato un ArrayList, contenente gli oggetti Edge, gli archi che sono collegati alla chiave (o nodo) contenenti un campo per la destinazione dell'arco e uno per il peso dell'arco.

*Node – campi*

• char c – char utilizzato come identificatore del nodo, può essere un lettera compresa tra 'A' e 'J'
• boolean visitato – true se il nodo è stato tolto defintivamente dalla priority queue
• double x – double utilizzato per memorizzare le coordinate grafiche del nodo sull'asse x
• double y – double utilizzato per memorizzare le coordinate grafiche del nodo sull'asse y
• int priority – priorità attuale del nodo nella priority queue
• boolean b – controllo che permette di vedere se un nodo deve essere ancora analizzato e quindi entrare nel cammino minimo
• int d – somma di ogni arco per arrivare al determinato nodo dall'inizio del grafo
• Node T – nodo precedentemente visitato

*Edge – campi*

• Integer weight – peso dell'arco che deve essere compreso tra 1 e 10
• Node destination – nodo di destinazione dell'arco

*Graph*

• HashMap< Node, ArrayList<Edge> > grafo – struttura dati scelta per la creazione del grafo

**Spotlight** **metodi:**
- *Costruttore:* il grafo ospita un massimo di 10 nodi, ognuno caratterizzato da un carattere alfabetico (da 'A' a 'J'). Il costruttore del grafo può creare un grafo vuoto oppure, se passato un intero corrispondente al numero dei nodi, un grafo con archi generati casualmente.
- *InsertNode:* quando vengono inseriti dei nuovi nodi, il grafo inserisce automaticamente un nodo a cui aggiunge la prima lettera dell'alfabeto disponibile. Se i nodi aggiunti hanno già raggiunto il numero massimo, un messaggio di errore avvertirà l'utente.

*Johnson*

• Graph grafo – utile per poter accedere ai metodi della classe grafo
• Queue<Node> Heap – coda con priorità

Dato un grafo pesato $G$ e un nodo di partenza $p \in G$, l'algoritmo di Johnson trova un cammino da $p$ ad $u$, per ogni nodo $u \in G$, tale che la somma dei pesi degli archi sia la più piccola possibile.

## Algoritmo di Johnson

```
shortestPath(Graph G, Node s)
PriorityQueue S = PriorityQueue(); S.insert(s, 0)
while not S.isEmpty() do
    u = S.deleteMin()
    b[u] = false
    foreach v ∈ G.adj(u) do
        if d[u] + G.w(u, v) < d[v] then
            if not b[v] then
                S.insert(v, d[u] + G.w(u, v))
                b[v] = true
            else
                S.decrease(v, d[u] + G.w(u, v))
            T[v] = u
            d[v] = d[u] + G.w(u, v)
return (T, d)
```

L'algoritmo inizia considerando un grafo *G* e un nodo *s*. Viene dichiarata una PriorityQueue *S* che conterrà i nodi valutabili per ogni passo dell'algoritmo. Il primo nodo *s* viene inserito nell'heap *S* con valore *d* uguale a 0 (essendo questo il primo nodo valutato, ha distanza dal primo nodo uguale a 0).

Finché il grafo non sarà vuoto verranno eseguite le seguenti operazioni:

1. Viene assegnato a *u* il nodo con priorità maggiore di tutti i nodi nell'heap. Il suo valore *b* si tramuta in false poiché questo controlla se un nodo deve essere ancora analizzato e quindi se può entrare nel cammino minimo.

2. Per tutti i nodi *v* adiacenti al nodo *u* (nel grafo *G*) preso in considerazione controlleremo se la distanza dal nodo di partenza *u* sommata alla distanza *d[u]* tra il nodo *u* e *v* è minore della distanza attuale di *d[v]* (impostata di default come ∞). In questa fase distinguiamo i nodi che devono ancora essere presi in esame da quelli gestiti dall'heap. Consideriamo i primi. Se la condizione è verificata allora inseriremo il nodo *v* nell'heap *S* tenendo conto della sua priorità che attualmente vale la distanza per arrivare al nodo *u* sommata alla distanza tra il nodo *u* e *v*. Il valore *b[v]* viene modificato in true poiché attualmente il nodo si trova in heap ed è in "attesa" di essere gestito. Se la condizione è falsa allora abbiamo trovato un modo migliore per raggiungere il nodo *v* così il suo valore viene aggiornato (decrementato) con il valore del nuovo percorso stabilito.

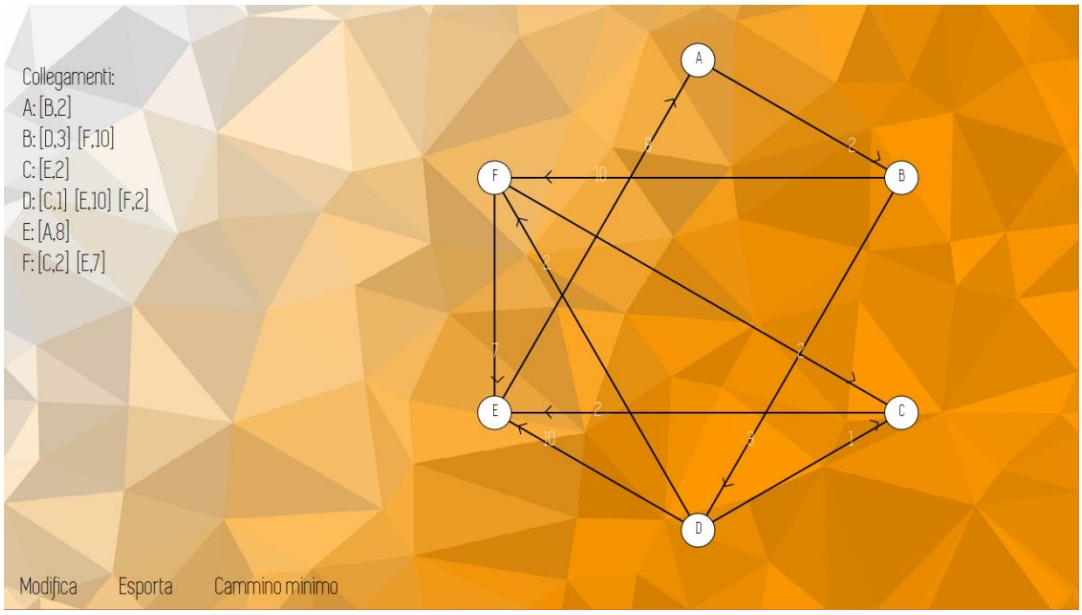
4. Imposto il valore *T* di *v* con il nodo *u* per tenere così traccia di quale nodo è stato visitato prima di *v.*

5. Aggiorno la distanza *d* di *v* con il nuovo valore di priorità contenuto anche nell'heap.

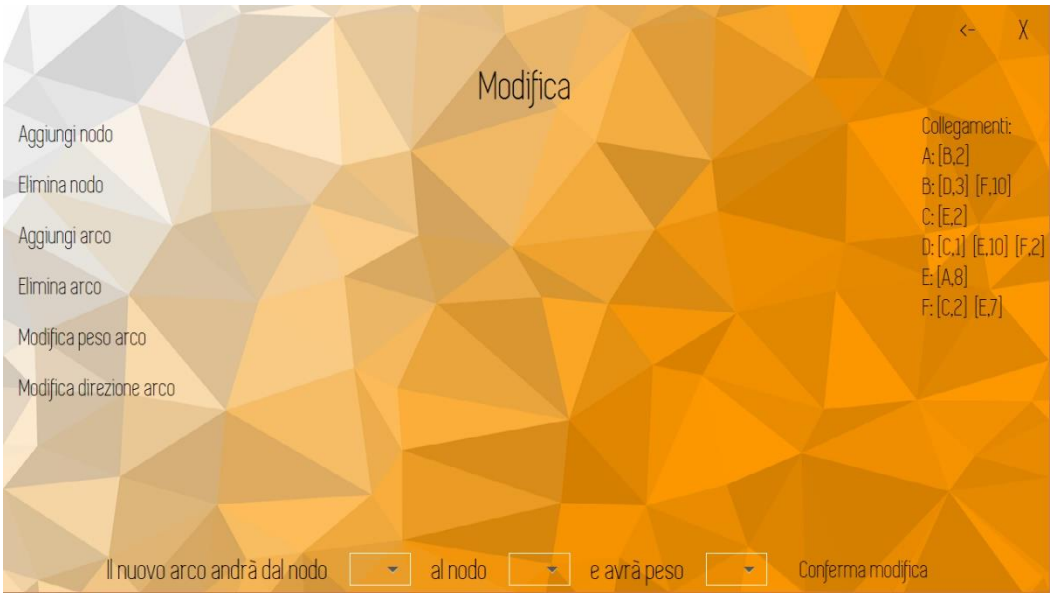
## Grafica

Per l'implementazione della grafica del progetto abbiamo optato per Java FXML, suggerito dal docente durante il corso. Inoltre, abbiamo anche utilizzato CSS per rendere il progetto visualmente piacevole. Font e sfondi sono stati scelti senza copyright.

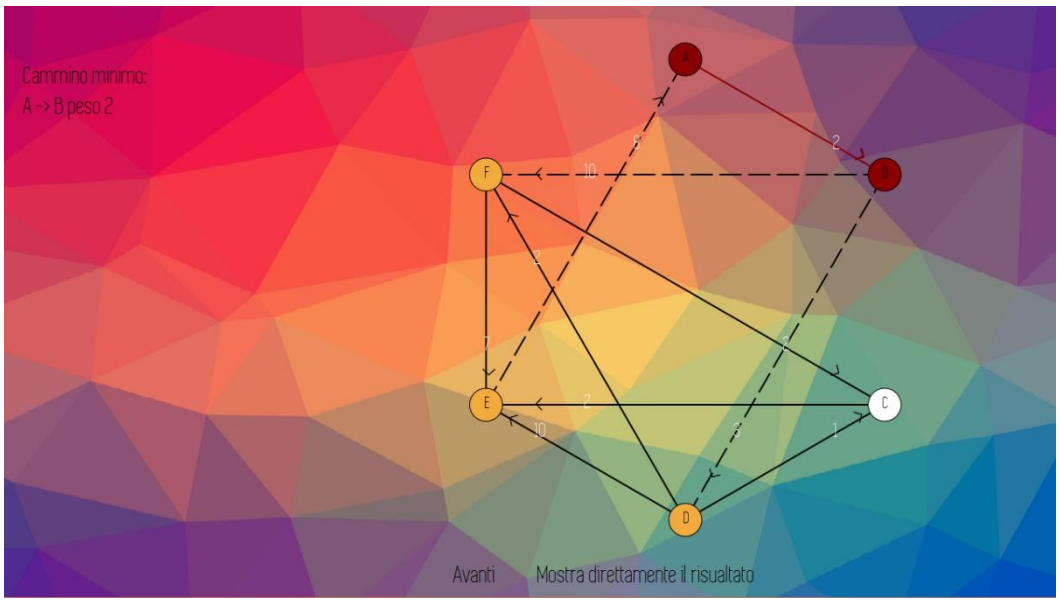

L'utente può scegliere se creare il grafo o importarlo (Scene 2 blu)




Schermata di visualizzazione del grafo




Editor del grafo




Esecuzione dell'algoritmo step-by-step

## Importazione e esportazione dei grafi

Nel software sono presenti delle impostazioni che ti permetteranno di esportare un qualsiasi grafo creato, è infatti possibile esportare sia un cammino minimo derivato da un grafo precedente che un semplice grafo creato a mano o generato casualmente. Riguardo l'importazione è possibile importare un grafo nella fase iniziale di progettazione dei grafi e solo in seguito si potranno apportare modifiche o determinare il grafo dei cammini minimi.

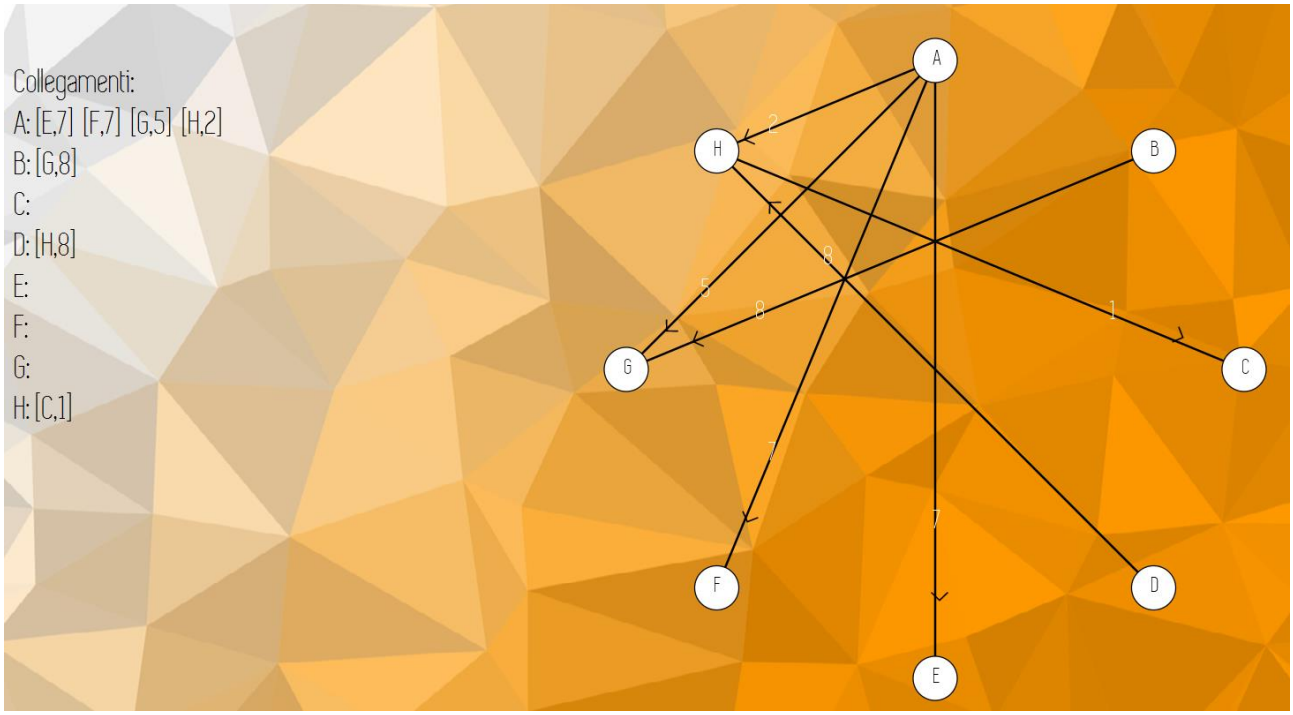
Quando si esporterà un grafo su un file .txt il risultato sarà simile al seguente:

```
A:E7,F7,G5,H2,.
B:G8,.
C:.
D:H8,.
E:.
F:.
G:.
H:C1,.

```

Per l'importazione ci sono le stesse caratteristiche che adesso descriveremo brevemente: Le lettere sulla sinistra sono tutti i nodi presenti nel grafo, dopo i ':' possiamo vedere tutti i nodi collegati al nodo in questione con il rispettivo peso dell'arco (tra 1 e 10), dopo ogni ',' si può inserire un altro nodo oppure terminare l'elenco dei nodi collegati tramite un '.' e uno ' ' (SPAZIO). Lo spazio non è visibile ma è fondamentale per far interpretare al programma il grafo da noi creato.

Il risultato finale di una corretta importazione sarà il seguente:




## Software e metodi di condivisione codice

**IntelliJ IDEA**

Abbiamo scelto un IDE comune a tutti per lo sviluppo del progetto per evitare problemi di compatibilità. La scelta è ricaduta su IntelliJ per le sue caratteristiche e funzionalità che ci hanno aiutato durante lo sviluppo del codice. Il suo utilizzo è stato possibile grazie all'account universitario.

**Dropbox**

Durante la fase di sviluppo abbiamo scelto di utilizzare Dropbox per la condivisione dei file, del codice e come archiviazione generale. Ad ogni aggiornamento sono inserite le nuove versioni del codice, cosicché una volta terminato il progetto abbiamo potuto monitorare l'intera evoluzione del codice.

**Codeshare**

Editor di testo open-source online, utilizzato per scambio di codice e programmazione simultanea.

**Discord**

Software per chiamate video e condivisione dello schermo che supporta chiamate tra due o più utenti.

## Fonti

https://stackoverflow.com/

https://docs.oracle.com/javase/8/

https://www.geeksforgeeks.org/

https://www.google.it/

https://it.wikipedia.org/

## Report Incontri

03/08/2018 – Brainstorming sulla realizzazione del progetto

Presenti: Martina Sosto, Andrea Vicenzi, Sara Vorabbi

10/08/2018 – Decisioni grafiche

Presenti: Martina Sosto, Andrea Vicenzi, Sara Vorabbi

23/08/2018 – Scelta delle strutture dati per l'implementazione del grafo

Presenti: Martina Sosto, Andrea Vicenzi, Sara Vorabbi

26/08/2018 – Scelte implementative sulla comunicazione fra grafo e parte grafica

Presenti: Martina Sosto, Sara Vorabbi

28/08/2018 – Unione codice grafo e parte grafica + aggiunta funzioni

Presenti: Martina Sosto, Sara Vorabbi

29/08/2018 – Unione codice grafo e parte grafica + aggiunta funzioni

Presenti: Martina Sosto, Sara Vorabbi

31/08/2018 – Implementazione indicatori direzionali grafo

Presenti: Martina Sosto, Sara Vorabbi

01/09/2018 – Implementazione algoritmo Johnson + priority queue + discussione a proposito dei file

Presenti: Martina Sosto, Andrea Vicenzi, Sara Vorabbi

02/09/2018 – Implementazione algoritmo Johnson + correzione bug

Presenti: Martina Sosto, Sara Vorabbi

03/09/2018 – Perfezionamento parte grafica + correzione bug + importazione/esportazione grafo e algoritmo su file .txt + stesura documenti

Presenti: Martina Sosto, Andrea Vicenzi, Sara Vorabbi

07/01/2019 – Correzione di eventuali errori riguardanti esportazione e importazione dei file e correzioni riguardanti l'organizzazione della struttura dell'heap

Presenti: Martina Sosto, Sara Vorabbi

## Monte ore

### Martina Sosto:

- 1h studio costruzione parte grafica progetto
- 2h inizio creazione contenuti grafici (prime tre scene con varie selezioni)
- 1h studio utilizzo FileChooser e ricerca Background
- 3h rifiniture grafiche (rimossa barra in alto finestra, aggiunto CSS per bottoni trasparenti) e avanzamento costruzione scene
- 4h ricerca sulla creazione del grafico e aggiunta schermata di modifiche del grafo
- 3h aggiunta delle scelte per la modifica del grafo (menu a tendina, textfield ed esportazione del file), creazione schermate di visualizzazione dell'algoritmo
- 3h correzione vari problemi legati alla grafica e studio della costruzione del grafo
- 5h creazione del grafo e ricerche correlate
- 5h creazione interazione tra GUI e codice di modifica del grafo
- 7h correzione visualizzazione grafo e risoluzione di bug vari
- 9h creazione funzione di esportazione di grafo su file+ modifiche grafiche+ studio e inizio implementazione algoritmo
- 7 ½h Implementazione algoritmo Johnson + correzione bug
- 6 ½h Stampa algoritmo Johnson + creazioni funzioni per l'esportazione del grafo
- 5h correzione esportazione del cammino minimo + importazione del file
- 2h correzione errori sull'importazione dei file
- 6h implementazione importazione e esportazione dei grafi sui file, correzione valori heap e documentazione

**TOTALE ORE IMPIEGATE:** 70h

### Andrea Vicenzi:

**TOTALE ORE IMPIEGATE:**

### Sara Vorabbi:

- 2h Studio delle strutture da utilizzare nel progetto(code con priorità e heap)
- 3h Ricerca su strutture dati

- 1h30 Ricerca su strutture dati
- 2h Ricerca e implementazione strutture dati
- 1h Implementazione classi
- 2h Scrittura funzioni grafo
- 2h Scrittura funzioni grafo
- 1h Modifica parametri passati funzioni grafo (input)
- 1h30 Modifica funzioni per modifica peso ed eliminazione arco
- 10h Correzione funzioni archi random, inverti direzione + fix bug + modifica valori di ritorno (boolean) + interfaccia
- 3h Assemblaggio a codice della parte grafica + fix bug + creazione nuove funzioni per parte grafica
- 1h Scrittura funzioni di accesso al grafo per esportazione su file
- 5h Correzione visualizzazione grafo
- 9h Studio e inizio implementazione algoritmo Johnson
- 7h30 Implementazione algoritmo Johnson + correzione bug
- 5h Scrittura documenti + fix bug
- 4h30 Implementazione heap e debugging
- 6h implementazione importazione e esportazione dei grafi sui file, esportazione cammino minimo, correzione valori heap e documentazione

**TOTALE ORE IMPIEGATE:** 66 ½h