Ponti Federico, 817395, f.ponti2@campus.unimib.it

Progetto parte 1: Grafo orientato

La prima parte consiste nella creazione di una classe templata che rappresenti un Grafo orientato.
Come membri della classe abbiamo un puntatore a uno o più Node, struct interna costituita da un *name* di tipo generico e un puntatore a una riga di *Matrix* (puntatore implementato inizialmente ma poi non utilizzato nel progetto) e *Size*, int che rappresenta il numero di nodi presenti.
Come altro membro abbiamo Matrix, la Matrice di adiacenza vera e propria che rappresenta i collegamenti - archi - tra un nodo e l'altro. La matrice avrà grandezza pari a *Size* x *Size*, con righe e colonne che rappresentano i vari nodi.
Per inizializzare un oggetto bisogna passare come parametri un int che rappresenta la *size* e un puntatore a uno o più elementi di tipo generico.

Dopo i metodi fondamentali della classe (costruttore che inizializza i membri, distruttore, copy constructor e operatore di assegnamento) sono presenti i quattro metodi per aggiungere/rimuovere rispettivamente un nodo e un arco.

Per aggiungere un nodo il metodo *addNode* crea un oggetto node a partire dal parametro di tipo generico e aggiunge una riga e una colonna alla matrice di adiacenza. Viene infine anche aggiornato *Size*. Nel caso esistesse già un nodo con lo stesso nome viene lanciato un errore.
Per rimuoverne uno, invece, *removeNode* cerca il nodo con lo stesso nome del parametro generico passato e, in caso questo esista, rimuove il node dall'array membro della classe ed elimina le rispettive riga e colonna - tramite indice.
Per fare ciò viene creata una matrice temporanea (con un *size* di più o meno uno in base al metodo chiamato) con i dati presi dell'originale più quelli del node da aggiungere o meno quello del node da rimuovere.

Per aggiungere un arco vengono passati il nome del nodo d'origine e quello di destinazione, vengono cercati nell'array membro della classe e, se trovati, viene posto il rispettivo bool nella matrice di adiacenza a true. La ricerca avviene tramite indice preso dall'array.
Per rimuovere un arco il procedimento è simile, con la sola differenza che il valore della matrice di adiacenza viene posto a false.
Nel caso aggiungendo/rimuovendo archi e nodi vengano passati come parametri nodi non esistenti o già esistenti vengono lanciati errori tramite l'espressione *throw*.

Abbiamo poi i metodi *exist* e *hasEdge* per sapere se, rispettivamente, un nodo esiste e se un arco tra due nodi esiste (e quindi il rispettivo valore nella matrice è posto a true).
Per la prima funzione viene cercato il nodo nell'array membro e, se esiste, viene restituito true.
Per cercare l'arco vengono cercati i nodi di origine e destinazione e, se esistono, viene controllato il valore nella matrice di adiacenza, altrimenti viene lanciato un errore.

I metodi *begin* e *end* servono per istanziare un const Iterator, classe implementata dentro graph.

Ci sono, infine, metodi di supporto come Swap o overload di operatori.
Nel file main.cpp, oltre a vari test con tipi base, è presente una struct custom chiamata *obj_test* per testare la classe con dei tipi custom.
Il file inoltre contiene la funzione *test_iterator* per testare un iteratore di graph.

Progetto parte 2: Sudoku

Questa parte del progetto usa un algoritmo ricorsivo con backtracking per risolvere in modo automatico un Sudoku.

Il metodo ricorsivo è *solve_sudoku* che controlla se la cella del Sudoku è piena e, in caso negativo, prova a inserire un numero. Controlla se il numero può essere inserito tramite il metodo *isSafe* (che verifica che non ci siano altri numeri uguali sulla stessa riga/colonna e "sezione") e in caso positivo richiama se stesso ricorsivamente passando il nuovo indice per la colonna successiva.
All'inizio del metodo, se l'indice della colonna supera quello massimo questo viene impostato a zero e viene aumentato di uno l'indice relativo alla riga.
Se si arriva al caso base, ovvero quando tutte le celle sono state riempite, viene ritornato il valore true che conferma la riuscita dell'operazione.

Per scrivere e leggere il valore delle celle viene usata una variabile membro della classe chiamata *grid*, che è una matrice di puntatori alle celle mostrate nella finestra dalla .ui.

Nel caso non si arrivi a una soluzione o nel caso i valori iniziali delle celle non fossero corretti (la funzione *thereAreInvalidCharacters* controlla che l'utente non abbia inserito caratteri in posizioni non consentite) viene mostrato un messaggio d'errore tramite la funzione *showError* che riempie la label di errore nella ui e impedisce l'ulteriore modifica delle celle.

Per mostrare all'utente la soluzione step-by-step viene applicato lo stesso algoritmo di risoluzione spiegato precedentemente ma applicato a una matrice momentanea. Una volta trovata la soluzione viene presa la cella da riempire e viene settato il valore con gli stessi indici presente nella matrice momentanea.
Per fare ciò la classe contiene un overload delle funzioni *solve_sudoku* e *isSafe* che modifica la grid membro della classe o una temporanea passata tramite puntatore.

Nella parte di ui viene applicato il metodo *setfixedSize* per non permettere il cambiamento di dimensione della finestra e sono presenti tre pulsanti che, tramite gli slot della classe *MainWindow*, azionano le funzioni rispettivamente per trovare la soluzione, fare lo step-by-step, o azzerrare il contenuto della griglia.
Viene inoltre applicato un validator alle celle che consente di inserire solo numeri da 1 a 9.