

SPaths

Generato da Doxygen 1.8.1.2

Mar 15 Lug 2014 13:53:09

Indice

1	SPaths	1
2	Indice dei tipi composti	3
2.1	Elenco dei tipi composti	3
3	Indice dei file	5
3.1	Elenco dei file	5
4	Documentazione delle classi	7
4.1	Riferimenti per la struct arco	7
4.1.1	Descrizione dettagliata	7
4.1.2	Documentazione dei membri dato	7
4.1.2.1	id_nodo_adj	7
4.1.2.2	peso	7
4.2	Riferimenti per la struct grafo	7
4.2.1	Descrizione dettagliata	8
4.2.2	Documentazione dei membri dato	8
4.2.2.1	lista_nodi	8
4.2.2.2	num_nodi	8
4.3	Riferimenti per la struct nodo	8
4.3.1	Descrizione dettagliata	8
4.3.2	Documentazione dei membri dato	8
4.3.2.1	id	8
4.3.2.2	lista_adiacenze	9
5	Documentazione dei file	11
5.1	Riferimenti per il file disegna_grafo.cc	11
5.1.1	Descrizione dettagliata	11
5.1.2	Documentazione delle funzioni	11
5.1.2.1	disegna_arco	11
5.1.2.2	disegna_grafo	12
5.1.2.3	disegna_percorso	12
5.2	Riferimenti per il file disegna_grafo.h	12

5.2.1	Descrizione dettagliata	13
5.2.2	Documentazione delle funzioni	13
5.2.2.1	disegna_arco	13
5.2.2.2	disegna_grafo	13
5.2.2.3	disegna_percorso	13
5.3	Riferimenti per il file operazioni_file.cc	14
5.3.1	Descrizione dettagliata	14
5.3.2	Documentazione delle funzioni	14
5.3.2.1	carica_da_file	14
5.3.2.2	salva_su_file	14
5.4	Riferimenti per il file operazioni_file.h	15
5.4.1	Descrizione dettagliata	15
5.4.2	Documentazione delle funzioni	15
5.4.2.1	carica_da_file	15
5.4.2.2	salva_su_file	16
5.5	Riferimenti per il file operazioni_grafo.cc	16
5.5.1	Descrizione dettagliata	16
5.5.2	Documentazione delle funzioni	17
5.5.2.1	bellman_ford	17
5.5.2.2	distruggi_grafo	17
5.5.2.3	genera_grafo_casuale	17
5.5.2.4	stampa_adiacenze	18
5.6	Riferimenti per il file operazioni_grafo.h	18
5.6.1	Descrizione dettagliata	18
5.6.2	Documentazione delle funzioni	18
5.6.2.1	bellman_ford	18
5.6.2.2	distruggi_grafo	19
5.6.2.3	genera_grafo_casuale	19
5.6.2.4	stampa_adiacenze	19
5.7	Riferimenti per il file SPaths.cc	19
5.7.1	Descrizione dettagliata	21
5.7.2	Documentazione delle funzioni	21
5.7.2.1	crea_grafo_handler	21
5.7.2.2	disegna_percorso_handler	21
5.7.2.3	draw_handler	21
5.7.2.4	hide_main_window	21
5.7.2.5	load_handler	21
5.7.2.6	main	21
5.7.2.7	save_handler	22
5.8	Riferimenti per il file strutt_dati.h	22

5.8.1	Descrizione dettagliata	22
5.8.2	Documentazione delle definizioni	22
5.8.2.1	D1	22
5.8.2.2	D2	23
5.8.2.3	D4	23
5.8.2.4	D8	23

Capitolo 1

SPaths

Questo programma genera grafi casuali orientati debolmente connessi, li disegna tramite interfaccia grafica e visualizza il percorso più breve tra 2 nodi di esso, se esiste.

Si invoca da linea di comando senza alcun parametro aggiuntionale:

- `./SPaths`

Il grafo orientato è implementato mediante liste di adiacenza.

La ricerca del percorso più breve è effettuata con l'algoritmo di Bellman-Ford (vedi [bellman_ford](#)).

Moduli

- **SPaths.cc:**
contiene la funzione main, necessaria per l'avvio del programma e gli handler per l'interfaccia grafica.
Utilizza [strutt_dati.h](#), [operazioni_grafo.h](#), [disegna_grafo.h](#), [operazioni_file.h](#)
- **strutt_dati.h:**
contiene le strutture dati utilizzate da tutti i moduli.
- **operazioni_grafo.cc:**
contiene le funzioni utilizzate dal programma per lavorare su tipi di dato grafo.
Utilizza [strutt_dati.h](#) e [operazioni_grafo.h](#)
- **disegna_grafo.cc:**
contiene le funzioni utilizzate dall'interfaccia grafica per disegnare il grafo ed i percorsi minimi.
Utilizza [strutt_dati.h](#), [disegna_grafo.h](#) e [operazioni_grafo.h](#)
- **operazioni_file.cc:**
contiene le funzioni utilizzate per salvare il grafo corrente su file e per caricarlo successivamente.
Utilizza [strutt_dati.h](#)

Si veda anche

[strutt_dati.h](#) Per vedere come è stato implementato il [grafo](#)

Autore

Federico Terzi

Versione

1.0

Capitolo 2

Indice dei tipi composti

2.1 Elenco dei tipi composti

Queste sono le classi, le struct, le union e le interfacce con una loro breve descrizione:

arco	Struttura di ogni arco	7
grafo	Struttura generale del grafo	7
nodo	Struttura di ogni nodo	8

Capitolo 3

Indice dei file

3.1 Elenco dei file

Questo è un elenco dei file documentati con una loro breve descrizione:

disegna_grafo.cc	11
disegna_grafo.h	12
operazioni_file.cc	14
operazioni_file.h	15
operazioni_grafo.cc	16
operazioni_grafo.h	18
SPaths.cc	19
strutt_dati.h	22

Capitolo 4

Documentazione delle classi

4.1 Riferimenti per la struct arco

Struttura di ogni arco.

```
#include <strutt_dati.h>
```

Attributi pubblici

- int [id_nodo_adj](#)
- int [peso](#)

4.1.1 Descrizione dettagliata

Struttura di ogni arco.

La struttura di ogni arco contiene semplicemente l'id del nodo a cui fa riferimento l'arco, e il peso di tale arco, rappresentato tramite valore intero. Da notare che il peso può essere anche negativo.

4.1.2 Documentazione dei membri dato

4.1.2.1 int arco::id_nodo_adj

Identificatore del nodo a cui punta l'arco

4.1.2.2 int arco::peso

Peso dell'arco

La documentazione per questa struct è stata generata a partire dal seguente file:

- [strutt_dati.h](#)

4.2 Riferimenti per la struct grafo

Struttura generale del grafo.

```
#include <strutt_dati.h>
```

Attributi pubblici

- int [num_nodi](#)
- GList * [lista_nodi](#)

4.2.1 Descrizione dettagliata

Struttura generale del grafo.

Grazie a questa struttura è possibile rappresentare un grafo diretto e pesato. E' formato da una lista doppia implementata nella libreria GLib, che fa riferimento a valori di tipo [nodo](#) e dal numero di nodi presenti nel grafo.

4.2.2 Documentazione dei membri dato

4.2.2.1 GList* grafo::lista_nodi

Lista dei nodi presenti nel grafo

4.2.2.2 int grafo::num_nodi

Numero di nodi presenti nel grafo

La documentazione per questa struct è stata generata a partire dal seguente file:

- [strutt_dati.h](#)

4.3 Riferimenti per la struct nodo

Struttura di ogni nodo.

```
#include <strutt_dati.h>
```

Attributi pubblici

- int [id](#)
- GList * [lista_adiacenze](#)

4.3.1 Descrizione dettagliata

Struttura di ogni nodo.

Ogni nodo contiene l'id di riferimento, rappresentato tramite un valore intero, e la propria lista di adiacenze, rappresentata tramite lista doppia, implementata nella libreria GLib. La lista di adiacenze di ogni nodo contiene dei puntatori a dati di tipo [arco](#)

4.3.2 Documentazione dei membri dato

4.3.2.1 int nodo::id

Identificatore del nodo

4.3.2.2 GList* nodo::lista_adiacenze

Lista di adiacenze del nodo

La documentazione per questa struct è stata generata a partire dal seguente file:

- [strutt_dati.h](#)

Capitolo 5

Documentazione dei file

5.1 Riferimenti per il file `disegna_grafo.cc`

```
#include <iostream>
#include <cstring>
#include <ctime>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include <limits>
#include <sstream>
#include <fstream>
#include <glib.h>
#include <gtk/gtk.h>
#include "strutt_dati.h"
#include "operazioni_grafo.h"
#include "disegna_grafo.h"
```

Funzioni

- void `disegna_arco` (cairo_t *cr, quint startX, quint startY, quint endX, quint endY)
Disegna un arco all'interno di un cairo context.
- void `disegna_grafo` (GtkWidget *widget, cairo_t *cr, const `grafo` &G, bool stampa_pesi)
Disegna un grafo all'interno di un cairo context.
- void `disegna_percorso` (cairo_t *cr, const `grafo` &G, const int sorgente, const int destinazione)
Disegna il percorso più breve per arrivare da un nodo a un altro nodo.

5.1.1 Descrizione dettagliata

Questo file contiene le funzioni utilizzate dal programma per disegnare sulla DrawingArea un grafo ed eventualmente un percorso richiesto. Utilizza 5 variabili globali con internal linkage.

5.1.2 Documentazione delle funzioni

5.1.2.1 void `disegna_arco` (cairo_t * cr, quint *startX*, quint *startY*, quint *endX*, quint *endY*)

Disegna un arco all'interno di un cairo context.

Questa funzione disegna una linea, partendo dai due punti di partenza e arrivando ai due punti di fine, definiti nei parametri con coordinate X e Y. Disegna inoltre un triangolo alla fine della linea, in modo da rappresentare graficamente una freccia.

Parametri

<i>cr</i>	Il cairo context su cui disegnare la freccia
<i>startX</i>	Le coordinate X del punto di partenza
<i>startY</i>	Le coordinate Y del punto di partenza
<i>endX</i>	Le coordinate X del punto di arrivo
<i>endY</i>	Le coordinate Y del punto di arrivo

5.1.2.2 void `disegna_grafo` (GtkWidget * *widget*, cairo_t * *cr*, const grafo & *G*, bool *stampa_pesi*)

Disegna un grafo all'interno di un cairo context.

Calcola l'altezza e la larghezza della DrawingArea passata come parametro per avere il raggio del cerchio inscritto all'interno. Questo cerchio non verrà disegnato ma sarà usato come base per disegnare i nodi in modo che siano equidistanti tra di loro e abbiano una posizione univoca, facilmente calcolabile mediante `sin()` e `cos()`. Si disegnano poi i nodi sul perimetro del cerchio calcolato in precedenza e gli archi, utilizzando la funzione [disegna_arco](#). Se il parametro `stampa_pesi` è TRUE, allora vengono scritti i pesi sopra ogni arco. Per fare in modo che i pesi non si sovrascrivano, il peso viene scritto a 3/4 dell'arco e non a metà. Infine su ogni nodo viene scritto l'id corrispondente.

Parametri

<i>widget</i>	Il widget sul quale disegnare il grafo, che sarà di tipo DrawingArea
<i>cr</i>	Il cairo context su cui disegnare il grafo
<i>G</i>	Il grafo da disegnare
<i>stampa_pesi</i>	Booleano settato a TRUE se devono essere scritti i pesi sugli archi

5.1.2.3 void `disegna_percorso` (cairo_t * *cr*, const grafo & *G*, const int *sorgente*, const int *destinazione*)

Disegna il percorso più breve per arrivare da un nodo a un altro nodo.

Utilizza la funzione [bellman_ford](#) per trovare i cammini minimi a partire dal nodo sorgente, specificato come parametro. Dopodichè verrà chiamata la funzione [disegna_arco](#) per tracciare gli archi, partendo dal nodo di destinazione finchè non si è arrivati dal nodo di partenza.

Parametri

<i>cr</i>	Il cairo context su cui tracciare il percorso
<i>G</i>	Il grafo su cui calcolare i cammini minimi
<i>sorgente</i>	Il nodo sorgente
<i>destinazione</i>	Il nodo destinazione

5.2 Riferimenti per il file `disegna_grafo.h`

```
#include <iostream>
#include <fstream>
```

Funzioni

- void [disegna_arco](#) (cairo_t **cr*, quint *startX*, quint *startY*, quint *endX*, quint *endY*)

Disegna un arco all'interno di un cairo context.

- void `disegna_grafo` (GtkWidget *widget, cairo_t *cr, const `grafo` &G, bool stampa_pesi)

Disegna un grafo all'interno di un cairo context.

- void `disegna_percorso` (cairo_t *cr, const `grafo` &G, const int sorgente, const int destinazione)

Disegna il percorso più breve per arrivare da un nodo a un altro nodo.

5.2.1 Descrizione dettagliata

Questo è l'header file di `disegna_grafo.cc`. Contiene le interfacce delle funzioni utilizzate dal programma per disegnare grafi all'interno di un cairo context.

5.2.2 Documentazione delle funzioni

5.2.2.1 void `disegna_arco` (cairo_t * cr, quint startX, quint startY, quint endX, quint endY)

Disegna un arco all'interno di un cairo context.

Questa funzione disegna una linea, partendo dai due punti di partenza e arrivando ai due punti di fine, definiti nei parametri con coordinate X e Y. Disegna inoltre un triangolo alla fine della linea, in modo da rappresentare graficamente una freccia.

Parametri

<code>cr</code>	Il cairo context su cui disegnare la freccia
<code>startX</code>	Le coordinate X del punto di partenza
<code>startY</code>	Le coordinate Y del punto di partenza
<code>endX</code>	Le coordinate X del punto di arrivo
<code>endY</code>	Le coordinate Y del punto di arrivo

5.2.2.2 void `disegna_grafo` (GtkWidget * widget, cairo_t * cr, const `grafo` & G, bool stampa_pesi)

Disegna un grafo all'interno di un cairo context.

Calcola l'altezza e la larghezza della `DrawingArea` passata come parametro per avere il raggio del cerchio inscritto all'interno. Questo cerchio non verrà disegnato ma sarà usato come base per disegnare i nodi in modo che siano equidistanti tra di loro e abbiano una posizione univoca, facilmente calcolabile mediante `sin()` e `cos()`. Si disegnano poi i nodi sul perimetro del cerchio calcolato in precedenza e gli archi, utilizzando la funzione `disegna_arco`. Se il parametro `stampa_pesi` è TRUE, allora vengono scritti i pesi sopra ogni arco. Per fare in modo che i pesi non si sovrascrivano, il peso viene scritto a 3/4 dell'arco e non a metà. Infine su ogni nodo viene scritto l'id corrispondente.

Parametri

<code>widget</code>	Il widget sul quale disegnare il grafo, che sarà di tipo <code>DrawingArea</code>
<code>cr</code>	Il cairo context su cui disegnare il grafo
<code>G</code>	Il grafo da disegnare
<code>stampa_pesi</code>	Booleano settato a TRUE se devono essere scritti i pesi sugli archi

5.2.2.3 void `disegna_percorso` (cairo_t * cr, const `grafo` & G, const int sorgente, const int destinazione)

Disegna il percorso più breve per arrivare da un nodo a un altro nodo.

Utilizza la funzione `bellman_ford` per trovare i cammini minimi a partire dal nodo sorgente, specificato come parametro. Dopodichè verrà chiamata la funzione `disegna_arco` per tracciare gli archi, partendo dal nodo di destinazione finchè non si è arrivati dal nodo di partenza.

Parametri

<i>cr</i>	Il cairo context su cui tracciare il percorso
<i>G</i>	Il grafo su cui calcolare i cammini minimi
<i>sorgente</i>	Il nodo sorgente
<i>destinazione</i>	Il nodo destinazione

5.3 Riferimenti per il file operazioni_file.cc

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <cassert>
#include <glib.h>
#include "strutt_dati.h"
```

Funzioni

- bool [salva_su_file](#) (const [grafo](#) &G, const char nomefile[])
Salva un grafo passato per parametro su un file di testo.
- bool [carica_da_file](#) ([grafo](#) &G, const char nomefile[])
Carica un grafo da file sul grafo passato per riferimento.

5.3.1 Descrizione dettagliata

Questo file contiene le funzioni utilizzate dal programma per salvare un grafo su file e caricarlo successivamente.

5.3.2 Documentazione delle funzioni

5.3.2.1 bool carica_da_file (grafo & G, const char nomefile[])

Carica un grafo da file sul grafo passato per riferimento.

Prova ad aprire il file passato per parametro in lettura e lo interpreta linea per linea. Il primo valore che deve leggere sarà l'id del nodo a cui deve puntare l'arco che esce dal nodo corrente (partendo da 0), mentre il valore successivo sarà il peso di quell'arco. Continua ad aggiungere archi sul nodo corrente finchè non trova il separatore "--—", quando lo trova passa ad aggiungere archi al nodo successivo.

Parametri

<i>G</i>	Il riferimento del grafo da caricare
<i>nomefile</i>	Il nome del file su cui è salvato il grafo

Restituisce

TRUE se l'operazione è andata a buon fine, FALSE altrimenti

5.3.2.2 bool salva_su_file (const grafo & G, const char nomefile[])

Salva un grafo passato per parametro su un file di testo.

Prova ad aprire il file passato per parametro in scrittura e scorre la lista dei nodi e la lista delle adiacenze per scriverle sul file. La prima linea rappresenterà il nodo a cui punta l'arco, mentre quella successiva rappresenterà il peso dell'arco. Il separatore "--—" delimita la fine degli archi per il nodo corrente.

Parametri

<i>G</i>	Il grafo da scrivere su file
<i>nomefile</i>	Il nome del file su cui scrivere il grafo

Restituisce

TRUE se l'operazione è andata a buon fine, FALSE altrimenti

5.4 Riferimenti per il file operazioni_file.h

```
#include <iostream>
#include <fstream>
```

Funzioni

- bool [salva_su_file](#) (const [grafo](#) &G, const char nomefile[])
Salva un grafo passato per parametro su un file di testo.
- bool [carica_da_file](#) ([grafo](#) &G, const char nomefile[])
Carica un grafo da file sul grafo passato per riferimento.

5.4.1 Descrizione dettagliata

Questo è l'header file di [operazioni_file.cc](#). Contiene le interfacce delle funzioni utilizzate dal programma per salvare un grafo su file e per caricarlo successivamente.

5.4.2 Documentazione delle funzioni

5.4.2.1 bool carica_da_file (grafo & G, const char nomefile[])

Carica un grafo da file sul grafo passato per riferimento.

Prova ad aprire il file passato per parametro in lettura e lo interpreta linea per linea. Il primo valore che deve leggere sarà l'id del nodo a cui deve puntare l'arco che esce dal nodo corrente (partendo da 0), mentre il valore successivo sarà il peso di quell'arco. Continua ad aggiungere archi sul nodo corrente finchè non trova il separatore "--—", quando lo trova passa ad aggiungere archi al nodo successivo.

Parametri

<i>G</i>	Il riferimento del grafo da caricare
<i>nomefile</i>	Il nome del file su cui è salvato il grafo

Restituisce

TRUE se l'operazione è andata a buon fine, FALSE altrimenti

5.4.2.2 bool salva_su_file (const grafo & G, const char nomefile[])

Salva un grafo passato per parametro su un file di testo.

Prova ad aprire il file passato per parametro in scrittura e scorre la lista dei nodi e la lista delle adiacenze per scriverle sul file. La prima linea rappresenterà il nodo a cui punta l'arco, mentre quella successiva rappresenterà il peso dell'arco. Il separatore "--—" delimita la fine degli archi per il nodo corrente.

Parametri

<i>G</i>	Il grafo da scrivere su file
<i>nomefile</i>	Il nome del file su cui scrivere il grafo

Restituisce

TRUE se l'operazione è andata a buon fine, FALSE altrimenti

5.5 Riferimenti per il file operazioni_grafo.cc

```
#include <iostream>
#include <cstring>
#include <ctime>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include <limits>
#include <sstream>
#include <fstream>
#include <glib.h>
#include <gtk/gtk.h>
#include "strutt_dati.h"
#include "operazioni_grafo.h"
```

Funzioni

- [grafo genera_grafo_casuale](#) (int nodi, int min_peso, int max_peso)
Genera un grafo casuale debolmente connesso.
- void [distruggi_grafo](#) (grafo &G)
Distrugge il grafo passato per riferimento, deallocando tutta la memoria da lui precedentemente allocata.
- void [bellman_ford](#) (const grafo &G, const int sorgente, int *&pred, int *&dist)
Calcola i cammini minimi di un grafo a partire da un determinato nodo, utilizzando l'algoritmo di Bellman-Ford.
- void [stampa_adiacenze](#) (const grafo &G, ostream &os)
Stampa le adiacenze del grafo in ingresso su un output stream passato per riferimento.

5.5.1 Descrizione dettagliata

Questo file contiene le funzioni utilizzate dal programma per lavorare sul tipo di dato grafo: creazione, distruzione, ricerca dei cammini minimi e stampa delle adiacenze.

5.5.2 Documentazione delle funzioni

5.5.2.1 void bellman_ford (const grafo & G, const int source, int *& pred, int *& dist)

Calcola i cammini minimi di un grafo a partire da un determinato nodo, utilizzando l'algoritmo di Bellman-Ford.

Vengono allocati in memoria due array delle dimensioni del numero di nodi del grafo che dovranno contenere (per ogni nodo) la distanza dal nodo sorgente (dist) e l'indice del nodo precedente (pred) per calcolare il cammino minimo. Le distanze vengono inizializzate al massimo per il valore intero, mentre gli indici dei nodi precedenti vengono inizializzati a -1. Si effettua il rilassamento degli archi per il num_nodi-1, ed un'ulteriore volta per controllare che non vi siano cicli negativi nel grafo. Se ci sono cicli negativi, gli array pred e dist saranno deallocati e i loro puntatori saranno settati a NULL. Se non esiste un cammino per un ipotetico nodo K, pred[K] sarà -1.

Parametri

<i>G</i>	Il grafo su cui calcolare i cammini minimi
<i>source</i>	Il nodo sorgente da cui partire per calcolare i cammini minimi
<i>pred</i>	Il puntatore all'array dei predecessori, deve essere NULL
<i>dist</i>	Il puntatore all'array delle distanze, deve essere NULL

5.5.2.2 void distruggi_grafo (grafo & G)

Distrugge il grafo passato per riferimento, deallocando tutta la memoria da lui precedentemente allocata.

Scorre tutta la lista dei nodi del grafo e per ogni nodo scorre la lista di adiacenze, deallocando tutti gli archi e tutti i nodi. Il numero dei nodi del grafo viene poi settato a 0 e il puntatore alla lista dei nodi viene settato a NULL.

Parametri

<i>G</i>	Il grafo da eliminare
----------	-----------------------

5.5.2.3 grafo genera_grafo_casuale (int nodi, int min_peso, int max_peso)

Genera un grafo casuale debolmente connesso.

Alloca il numero di nodi passato come parametro in memoria e li appende alla lista dei nodi del grafo, assegnando ad ognuno il proprio id ed inizializzando temporaneamente la loro lista di adiacenze a NULL. Dopodichè vengono create le adiacenze: per rendere il grafo connesso è stato deciso che esiste sempre l'arco che collega il nodo N con il nodo N+1. Inoltre ad ogni nodo viene aggiunto un numero casuale di archi che puntano ad altri nodi, scelti sempre casualmente.

Si veda anche

[strutt_dati.h](#) per avere più informazioni su come è strutturato il tipo di dato [grafo](#).

Parametri

<i>nodi</i>	Il numero di nodi del grafo da generare
<i>min_peso</i>	Il peso minimo degli archi
<i>max_peso</i>	Il peso massimo degli archi

Restituisce

Il grafo generato

5.5.2.4 void stampa_adiacenze (const grafo & G, ostream & os)

Stampa le adiacenze del grafo in ingresso su un output stream passato per riferimento.

Scorre tutta la lista dei nodi e per ogni nodo stampa la lista delle adiacenze sull'output stream passato per riferimento.

Parametri

<i>G</i>	Il grafo da stampare
<i>os</i>	L'output stream su cui stampare le adiacenze del grafo

5.6 Riferimenti per il file operazioni_grafo.h

```
#include <iostream>
#include <fstream>
```

Funzioni

- [grafo genera_grafo_casuale](#) (int nodi, int min_peso, int max_peso)
Genera un grafo casuale debolmente connesso.
- void [distruggi_grafo](#) (grafo &G)
Distrugge il grafo passato per riferimento, deallocando tutta la memoria da lui precedentemente allocata.
- void [bellman_ford](#) (const grafo &G, const int source, int *&pred, int *&dist)
Calcola i cammini minimi di un grafo a partire da un determinato nodo, utilizzando l'algoritmo di Bellman-Ford.
- void [stampa_adiacenze](#) (const grafo &G, ostream &os)
Stampa le adiacenze del grafo in ingresso su un output stream passato per riferimento.

5.6.1 Descrizione dettagliata

Questo è l'header file di [operazioni_grafo.cc](#). Contiene le interfacce delle funzioni utilizzate dal programma per lavorare su tipi di dato grafo.

5.6.2 Documentazione delle funzioni

5.6.2.1 void bellman_ford (const grafo & G, const int source, int *& pred, int *& dist)

Calcola i cammini minimi di un grafo a partire da un determinato nodo, utilizzando l'algoritmo di Bellman-Ford.

Vengono allocati in memoria due array delle dimensioni del numero di nodi del grafo che dovranno contenere (per ogni nodo) la distanza dal nodo sorgente (dist) e l'indice del nodo precedente (pred) per calcolare il cammino minimo. Le distanze vengono inizializzate al massimo per il valore intero, mentre gli indici dei nodi precedenti vengono inizializzati a -1. Si effettua il rilassamento degli archi per il num_nodi-1, ed un'ulteriore volta per controllare che non vi siano cicli negativi nel grafo. Se ci sono cicli negativi, gli array pred e dist saranno deallocati e i loro puntatori saranno settati a NULL. Se non esiste un cammino per un ipotetico nodo K, pred[K] sarà -1.

Parametri

<i>G</i>	Il grafo su cui calcolare i cammini minimi
<i>source</i>	Il nodo sorgente da cui partire per calcolare i cammini minimi
<i>pred</i>	Il puntatore all'array dei predecessori, deve essere NULL
<i>dist</i>	Il puntatore all'array delle distanze, deve essere NULL

5.6.2.2 void distruggi_grafo (grafo & G)

Distrugge il grafo passato per riferimento, deallocando tutta la memoria da lui precedentemente allocata.

Scorre tutta la lista dei nodi del grafo e per ogni nodo scorre la lista di adiacenze, deallocando tutti gli archi e tutti i nodi. Il numero dei nodi del grafo viene poi settato a 0 e il puntatore alla lista dei nodi viene settato a NULL.

Parametri

<i>G</i>	Il grafo da eliminare
----------	-----------------------

5.6.2.3 grafo genera_grafo_casuale (int nodi, int min_peso, int max_peso)

Genera un grafo casuale debolmente connesso.

Alloca il numero di nodi passato come parametro in memoria e li appende alla lista dei nodi del grafo, assegnando ad ognuno il proprio id ed inizializzando temporaneamente la loro lista di adiacenze a NULL. Dopodichè vengono create le adiacenze: per rendere il grafo connesso è stato deciso che esiste sempre l'arco che collega il nodo N con il nodo N+1. Inoltre ad ogni nodo viene aggiunto un numero casuale di archi che puntano ad altri nodi, scelti sempre casualmente.

Si veda anche

[strutt_dati.h](#) per avere più informazioni su come è strutturato il tipo di dato [grafo](#).

Parametri

<i>nodi</i>	Il numero di nodi del grafo da generare
<i>min_peso</i>	Il peso minimo degli archi
<i>max_peso</i>	Il peso massimo degli archi

Restituisce

Il grafo generato

5.6.2.4 void stampa_adiacenze (const grafo & G, ostream & os)

Stampa le adiacenze del grafo in ingresso su un output stream passato per riferimento.

Scorre tutta la lista dei nodi e per ogni nodo stampa la lista delle adiacenze sull'output stream passato per riferimento.

Parametri

<i>G</i>	Il grafo da stampare
<i>os</i>	L'output stream su cui stampare le adiacenze del grafo

5.7 Riferimenti per il file SPaths.cc

```
#include <iostream>
```

```
#include <cstring>
#include <ctime>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include <limits>
#include <sstream>
#include <glib.h>
#include <gtk/gtk.h>
#include "strutt_dati.h"
#include "operazioni_grafo.h"
#include "disegna_grafo.h"
#include "operazioni_file.h"
```

Funzioni

- void [hide_main_window](#) (GtkButton *button, gpointer user_data)
Handler utilizzato per chiudere le dialog a partire dai segnali dei bottoni.
- gboolean [handler_delete_event](#) (GtkWidget *widget, GdkEvent *event, gpointer user_data)
Handler utilizzato per chiudere il programma.
- gboolean [handler_dialog_delete_event](#) (GtkWidget *widget, GdkEvent *event, gpointer user_data)
Handler utilizzato dal dialog per impedire di distruggersi tramite pressione del tasto Esc.
- void [load_handler](#) (GtkWidget *widget, gpointer data)
Handler utilizzato per caricare il grafo dal file grafo.txt.
- void [save_handler](#) (GtkWidget *widget, gpointer data)
Handler utilizzato per salvare il grafo corrente sul file grafo.txt.
- void [crea_grafo_handler](#) (GtkWidget *widget, gpointer data)
Handler utilizzato dalla dialog1 per creare un grafo.
- void [disegna_percorso_handler](#) (GtkWidget *widget, gpointer data)
Disegna il percorso richiesto dalla dialog2.
- void [nuovo_grafo_handler](#) (GtkMenuItem *menuitem, gpointer user_data)
Handler che si occupa di aprire la dialog1 per la creazione di un nuovo grafo.
- void [elimina_grafo_handler](#) (GtkMenuItem *menuitem, gpointer user_data)
Handler che elimina il grafo tramite la funzione [distruggi_grafo](#) e resetta le informazioni sul percorso. Inoltre aggiorna la barra di stato e la textview con le adiacenze del grafo.
- void [trova_percorso_handler](#) (GtkMenuItem *menuitem, gpointer user_data)
Handler che si occupa di aprire la dialog2 per la ricerca di un percorso.
- void [nascondi_percorso_handler](#) (GtkMenuItem *menuitem, gpointer user_data)
Handler che resetta i valori utilizzati per disegnare il percorso, nascondendolo.
- void [mostra_adiacenze_handler](#) (GtkMenuItem *menuitem, gpointer user_data)
Handler che mostra la dialog contenente le adiacenze del grafo, scritte su una textview.
- gboolean [draw_handler](#) (GtkWidget *widget, cairo_t *cr, gpointer data)
Handler che si occupa di disegnare sia il grafo che il percorso desiderato sulla DrawingArea della finestra principale del programma.
- void [help_handler](#) (GtkWidget *widget, gpointer data)
Handler che apre un'istanza di firefox in background per mostrare la documentazione.
- int [main](#) (int argc, char *argv[])
Funzione principale del programma.

5.7.1 Descrizione dettagliata

Il file principale contenente la funzione [main](#) . Il file utilizza 6 variabili globali con internal linkage, che contengono informazioni sul grafo, sul builder utilizzato dall'interfaccia grafica e sul percorso da visualizzare. Inoltre sono presenti tutti gli handler necessari all'interfaccia grafica.

5.7.2 Documentazione delle funzioni

5.7.2.1 void crea_grafo_handler (GtkWidget * widget, gpointer data)

Handler utilizzato dalla dialog1 per creare un grafo.

Utilizza i valori degli spinbutton e la funzione [genera_grafo_casuale](#) per generare un grafo casuale, resetta le variabili globali relative al percorso e richiede l'aggiornamento della DrawingArea, della barra di stato e della textview contenente le adiacenze del grafo.

5.7.2.2 void disegna_percorso_handler (GtkWidget * widget, gpointer data)

Disegna il percorso richiesto dalla dialog2.

Prende in ingresso i nodi di partenza e di arrivo grazie agli SpinButton della dialog2, e utilizza la funzione [bellman_ford](#) per controllare che esista un percorso tra i due nodi selezionati e che non vi sia un ciclo negativo nel grafo. Successivamente chiede un refresh alla DrawingArea, e sarà il [draw_handler](#) che si occuperà di disegnare il percorso.

5.7.2.3 gboolean draw_handler (GtkWidget * widget, cairo_t * cr, gpointer data)

Handler che si occupa di disegnare sia il grafo che il percorso desiderato sulla DrawingArea della finestra principale del programma.

Controlla se il grafo è stato inizializzato e disegna il grafo mediante la funzione [disegna_grafo](#), inoltre se è stato richiesto di disegnare un percorso, chiama la funzione [disegna_percorso](#)

5.7.2.4 void hide_main_window (GtkWidget * button, gpointer user_data)

Handler utilizzato per chiudere le dialog a partire dai segnali dei bottoni.

Viene chiuso il top level del GtkWidget passato per parametro.

Parametri

<i>button</i>	Il GtkWidget che richiede la chiusura della propria finestra
---------------	--

5.7.2.5 void load_handler (GtkWidget * widget, gpointer data)

Handler utilizzato per caricare il grafo dal file grafo.txt.

Se è stato già creato un grafo, viene distrutto. Viene poi caricato il grafo con la funzione [carica_da_file](#) e vengono aggiornate la DrawingArea, la barra di stato e la textview con le informazioni sul grafo. Se non è possibile aprire il file in lettura, viene mostrata una dialog di errore.

5.7.2.6 int main (int argc, char * argv[])

Funzione principale del programma.

Setta i nodi iniziali del grafo a 0 ed inizializza l'interfaccia tramite un builder, utilizzando il file "interfaccia.glade" e connettendo i segnali.

Restituisce

Valore di ritorno del programma

5.7.2.7 void save_handler (GtkWidget * widget, gpointer data)

Handler utilizzato per salvare il grafo corrente sul file grafo.txt.

Se il grafo corrente è stato inizializzato (e quindi `G.num_nodi > 0`) viene salvato su file grazie alla funzione [salva_su_file](#). Se non è possibile aprire il file in scrittura o se il grafo non è stato creato, viene mostrata una dialog di errore.

5.8 Riferimenti per il file strutt_dati.h

```
#include <iostream>
#include <string.h>
#include <glib.h>
#include <cassert>
```

Composti

- struct [nodo](#)
Struttura di ogni nodo.
- struct [arco](#)
Struttura di ogni arco.
- struct [grafo](#)
Struttura generale del grafo.

Definizioni

- #define **NDEBUG**
- #define **DBG**(A, B)
- #define **D1**(a) DBG(1, a)
- #define **D2**(a) DBG(2, a)
- #define **D4**(a) DBG(4, a)
- #define **D8**(a) DBG(8, a)

5.8.1 Descrizione dettagliata

Header file con le dichiarazioni della struttura dati comune a tutto il programma. Contiene inoltre le macro per il debug. Il grafo è rappresentato con una lista di nodi, e ogni nodo contiene la propria lista di adiacenze.

5.8.2 Documentazione delle definizioni

5.8.2.1 #define D1(a) DBG(1, a)

Macro per il debug, visualizza le azioni richieste dall'utente tramite handler

5.8.2.2 `#define D2(a) DBG(2, a)`

Macro per il debug, visualizza le azioni relative alle operazioni sul grafo (creazione, distruzione, ricerca percorsi).

5.8.2.3 `#define D4(a) DBG(4, a)`

Macro per il debug, visualizza le azioni relative alle operazioni su file

5.8.2.4 `#define D8(a) DBG(8, a)`

Macro per il debug visualizza le azioni relative al disegno del grafo sulla `DrawingArea`

Indice analitico

- arco, [7](#)
 - id_nodo_adj, [7](#)
 - peso, [7](#)
- bellman_ford
 - operazioni_grafo.cc, [17](#)
 - operazioni_grafo.h, [18](#)
- carica_da_file
 - operazioni_file.cc, [14](#)
 - operazioni_file.h, [15](#)
- crea_grafo_handler
 - SPaths.cc, [21](#)
- D1
 - strutt_dati.h, [22](#)
- D2
 - strutt_dati.h, [22](#)
- D4
 - strutt_dati.h, [23](#)
- D8
 - strutt_dati.h, [23](#)
- disegna_arco
 - disegna_grafo.cc, [11](#)
 - disegna_grafo.h, [13](#)
- disegna_grafo
 - disegna_grafo.cc, [12](#)
 - disegna_grafo.h, [13](#)
- disegna_grafo.cc, [11](#)
 - disegna_arco, [11](#)
 - disegna_grafo, [12](#)
 - disegna_percorso, [12](#)
- disegna_grafo.h, [12](#)
 - disegna_arco, [13](#)
 - disegna_grafo, [13](#)
 - disegna_percorso, [13](#)
- disegna_percorso
 - disegna_grafo.cc, [12](#)
 - disegna_grafo.h, [13](#)
- disegna_percorso_handler
 - SPaths.cc, [21](#)
- distruggi_grafo
 - operazioni_grafo.cc, [17](#)
 - operazioni_grafo.h, [18](#)
- draw_handler
 - SPaths.cc, [21](#)
- genera_grafo_casuale
 - operazioni_grafo.cc, [17](#)
 - operazioni_grafo.h, [19](#)
- grafo, [7](#)
 - lista_nodi, [8](#)
 - num_nodi, [8](#)
- hide_main_window
 - SPaths.cc, [21](#)
- id
 - nodo, [8](#)
- id_nodo_adj
 - arco, [7](#)
- lista_adiacenze
 - nodo, [8](#)
- lista_nodi
 - grafo, [8](#)
- load_handler
 - SPaths.cc, [21](#)
- main
 - SPaths.cc, [21](#)
- nodo, [8](#)
 - id, [8](#)
 - lista_adiacenze, [8](#)
- num_nodi
 - grafo, [8](#)
- operazioni_file.cc, [14](#)
 - carica_da_file, [14](#)
 - salva_su_file, [14](#)
- operazioni_file.h, [15](#)
 - carica_da_file, [15](#)
 - salva_su_file, [16](#)
- operazioni_grafo.cc, [16](#)
 - bellman_ford, [17](#)
 - distruggi_grafo, [17](#)
 - genera_grafo_casuale, [17](#)
 - stampa_adiacenze, [17](#)
- operazioni_grafo.h, [18](#)
 - bellman_ford, [18](#)
 - distruggi_grafo, [18](#)
 - genera_grafo_casuale, [19](#)
 - stampa_adiacenze, [19](#)
- peso
 - arco, [7](#)
- SPaths.cc, [19](#)
 - crea_grafo_handler, [21](#)
 - disegna_percorso_handler, [21](#)

- draw_handler, [21](#)
- hide_main_window, [21](#)
- load_handler, [21](#)
- main, [21](#)
- save_handler, [22](#)
- salva_su_file
 - operazioni_file.cc, [14](#)
 - operazioni_file.h, [16](#)
- save_handler
 - SPaths.cc, [22](#)
- stampa_adiacenze
 - operazioni_grafo.cc, [17](#)
 - operazioni_grafo.h, [19](#)
- strutt_dati.h, [22](#)
 - D1, [22](#)
 - D2, [22](#)
 - D4, [23](#)
 - D8, [23](#)