Ferdinando Del Trionfo 566/739 Paolo Selce 50/747
Gruppo 27

Quarto progetto

Analisi del problema:

Sono dati due file di testo. Il primo è il solito file con tag che è stato usato a varie riprese negli esercizi precedenti. Il secondo è un file di testo che determina delle regole di organizzazione gerarchica fra i tags. Pur lasciando libertà agli studenti su come questo file debba essere organizzato, alcune caratteristiche devono essere rispettate:
le regole devono avere una struttura che determini cosa contiene ogni tag;
ogni tag può contenere altri tag AND/OR lo stesso tag ricorsivamente AND/OR delle stringhe.

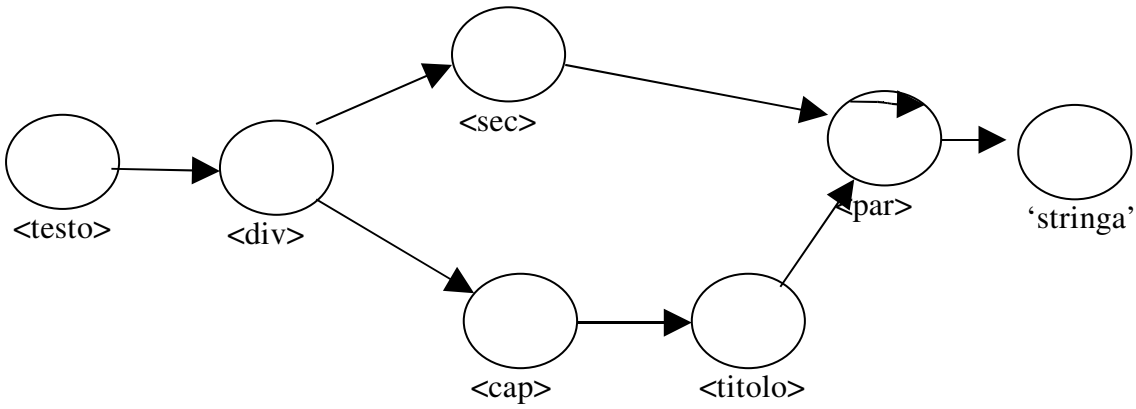
Esempio di possibili regole:

<testo> contiene
 contiene <sec> OR <cap>
<cap> contiene <titolo> AND <par>
<sec> contiene <par>
<par> contiene stringhe OR <par>

Le regole devono essere codificate con un sistema ben strutturato, il carattere di ritorno a capo o un carattere speciale definito da voi deve delimitare l estensione di ogni regola.

Le stringhe sono elementi terminali e hanno uno statuto speciale di cui discuteremo più avanti.

Generare un grafo (utilizzando strutture dinamiche quali ad esempio una lista di puntatori a liste) che rappresenti l insieme delle regole definite nel file di testo. Si consiglia lo studio e l impiego di una libreria per la gestione delle espressioni regolari per l analisi delle stringhe che formano le regole.
Esempio ricavato dalle regole sopra elencate.



Deve essere possibile modificare in ogni momento la struttura di questo grafo aggiungendo, cancellando o modificando una regola attraverso l inserimento di richieste da parte dell utente. Dopo le modifiche il grafo deve rimanere connesso, le regole che modificano il grafo non devono introdurre ambiguità nella struttura del grafo stesso.

Produrre una conversione del grafo basato su struttura dinamica in una matrice NxN statica e usare questa rappresentazione per stampare a video il grafo da menu. Provvedere nella documentazione a fornire una immagine come quella in figura per ogni set di regole utilizzato per testare l esercizio.

Analizzando tutti i possibili percorsi sul grafo si determini il percorso più breve per andare dal tag di start a un tag di tipo stringa. Se si implementa la parte facoltativa che segue, cancellare a caso uno dei tag sinistra-destra e calcolare il percorso minimo per andare dal simbolo di start ad una stringa e tornare a start per una via necessariamente diversa.

Facoltativo ma fortemente consigliato:

Ripetere la verifica di coerenza proposta nel terzo esercizio del documento con markup utilizzando contemporaneamente il grafo e lo stack.
Al grafo devono essere aggiunti archi inversi per ognuno degli archi previsti, in corrispondenza delle transizioni destra-sinistra si fa un push e in corrispondenza delle sinistra-destra si fa un pop. In questo modo sarete in grado di classificare con precisione gli eventuali errori di sintassi presenti nel documento con markup.

dettagli funzionali:
L' utente potra' scegliere le possibili scelte da un menu come: inserire il nome del file dei 2 file da elaborare inserire una regola,cancellare una regola.

Fase progettuale:

Vincoli progettuali
Strutture dati utilizzabili: array,stack,liste,grafi con tipo di memorizzazione : liste di adiacenza e matrici di adiacenza.
Riutilizzo massimo delle funzioni utilizzate nei progetti precedenti.

note progettuali:
Visto che le strutture dati principali vengono utilizzate spesso per ogni progetto, si e' pensato di implementarle in maniera piu astratta possibile, rappresentando il campo informazione di ogni struttura dati con un puntatore a void. Tale scelta ci permette di utilizzare le stesse strutture dati con le relative funzioni semplicemente facendo un cast al campo informazione.

Il progetto viene suddiviso come segue:

- ➔ Costruzione menu
- ➔ Caricamento,controllo e modifica delle regole
- ➔ Controllo Correttezza di un FILE (Secondo le regole)
- ➔ Calcolo percorso minimo
- ➔ Stampa a video regole

Fondamentalmente le parti interessanti del progetto sono legate al secondo modulo, perche' in tale modulo vengono utilizzate le funzioni classiche sui grafi e vengono fatte le scelte sui protocolli interni delle regole. Gli altri moduli sono meno interessanti perche' sostanzialmente sono funzioni gia implementate nei progetti precedenti; in tali pero vengono evidenziate le funzioni riutilizzate con eventuali modifiche .

(Riutilizzo del codice e della documentazione interna esterna dei progetti precedenti)

Analisi e descrizione del modulo Costruzione e stampa menu

La costruzione e la stampa del menu e' la parte di codice che serve all'utente per interfacciarsi con il programma. Infatti una delle richieste esplicite fatte dal cliente e' la possibilita di scegliere una delle seguenti opzioni: Verifica testo, stampa le i-esime parole,Unisci i-esime e k-esime parole. L'interfaccia grafica dovra' essere composta da un'intestazione, una sequenza di scelte, e la domanda di scelta; come di seguito:

MENU
1 Scelta1
2 Scelta2
Fai la tua scelta:

La soluzione ottimale sarebbe implementare tale menu di scelta con la possibilita di riutilizzarlo e di fare nuove aggiunte al menu in futuro. Tale scelta ci vincola a progettare la gestione del menu con file di appoggio in modo da poterli modificare.

sequenza naturale:
- Costruisci il menu
- Stampa menu

Analisi e descrizione del modulo Caricamento,controllo e modifica delle regole

Questo modulo fornisce gli strumenti per poter caricare controllare modificare le regole.

Tale modulo viene ulteriormente scomposto nelle seguenti sotto sezioni:

- Caricamento delle regole
- Controllo sintattico regole
- Modifica runtime delle regole
- Controllo grafo delle regole

Il caricamento delle regole prevede la lettura da file di tale regole e l'inserimento di tali in un grafo.
Il controllo delle regole prevede l'eventuale controllo sintattico secondo un protocollo interno ben definito.
Modifica runtime delle regole prevede durante l'esecuzione del programma la possibilita' di inserire nuove regole oppure di cancellarle. Ovviamente la cancellazione di tali regole si intende l'eliminazione dal grafo di eventuali nodi o archi.
Tale sezione fa un uso massiccio dei grafi quindi prevede un set di funzioni che sono descritte in dettaglio nella sezione strutture dati e routine fondamentali.

Analisi e descrizione del modulo controllo Correttezza dei un FILE (Secondo le regole)
Tale modulo riutilizza funzioni gia implementate nel progetto precedente, modificato per i grafi.
Esso prevede il controllo sintattico da un file di testo formato da TAG e TESTO in base alle regole date da file.

Analisi e descrizione del modulo calcolo percorso minimo
Questo modulo prevede il calcolo del percorso minimo di un grafo.
Il grafo ovviamente sara quello costruito nel secondo modulo di regole.
Tale modulo calcola il percorso minimo dal TAG di start a una qualsiasi stringa di stop e calcola anche il percorso minimo inverso da una stringa al TAG di start.

Analisi e descrizione del modulo Stampa a video regole
Questo modulo prevede la stampa del grafo contenente le regole rappresentato come matrice di adiacenza.
Questo modulo prevede prima la trasformazione della rappresentazione del grafo da liste di adiacenza in matrice di adiacenza.

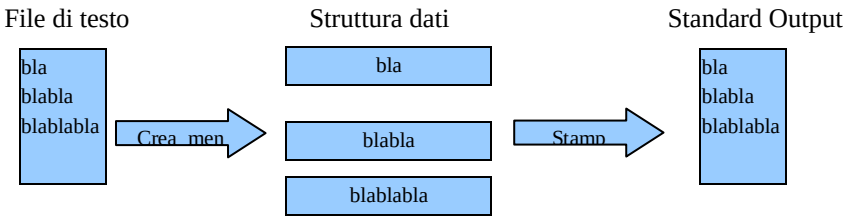
Cotruzione e stampa del menu (Interfaccia grafica)

(Riutilizzo del codice e della documentazione interna esterna del primo progetto)

L'interfaccia grafica visualizzata all'utente all'avvio del programma sara' formata da un menu' composto dalle seguenti selezioni :

1 Carica file regole
2 Aggiungi una regola
3 Elimina una regola (NODO)
4 - Elimina una regola (ARCO)
5 Controlla correttezza da file
6 Calcola percorso minimo
7 Stampa a video regole
8 Uscita

L'idea di creare un menu riutilizzabile e aggiornabile e' la seguente: costruire una funzione che prenda da un file di testo (con una sintassi e semantica prestabilita) un Menu. Tale funzione riempira' una struttura dati contenente il menu. Tale menu potrebbe essere formattato visualizzato a proprio piacimento infatti parallelamente a questa funzione ce ne dovra' essere un' altra la quale stampera' tale menu.



Pascal Like

La funzione Crea_menu(file_di_testo) prende come parametro di input una stringa che rappresenta il nome del file di testo; tale file di testo contiene la struttura del menu.
Sintassi e Semantica del file:
Il file e' composto da una parte d' intestazione, da una parte di scelte, e dall'ultima parte di domanda di selezione.
la prima riga rappresenta l'intestazione del menu.
Le righe restanti rappresentano le scelte del menu.
L'ultima riga rappresenta la domanda da visualizzare per le scelte.
valori ritornati:
La funzione ritorna un array di stringhe il quale conterra' al primo posto il nome del menu e nei

restanti posti prima dell'ultimo le relative scelte; e nell'ultimo posto la domanda di scelta.
Inoltre la funzione ritorna il numero di elementi inseriti nell'array.

esempio:

File_di_testo preso in input

Menu
1-Scelta1
2-Scelta2
fai una scelta:

Array ritornato

L'array ritornato verra' stampato a video dalla funzione stampa_menu(array,elementi) che prende in input l'array ritornato dalla funzione e il numero di elementi crea_menu.

PROCEDURE e PARAMETRI
Crea_Menu(INPUT:STRING file_di_testo OUTPUT:STRING ARRAY array , INT elem)
Stampa_Menu(INPUT: STRING ARRAY array , INT elem OUTPUT:)

Dettagli implementativi (orientato a C)
Le funzioni di creazione e stampa menu' sono implementate nel file funzioni_menu.c e definite nel file funzioni_menu.h .
La funzione Crea_Menu non puo costruire menu piu lunghi di 9 scelte per evidenti motivi.
La funzione Stampa_Menu stampa a video senza formattazioni speciali.
prototipi di funzioni:
char ** Crea_Menu(char* ,int);
void Stampa_Menu(char **,int);
I dettagli delle parti implementate si possono trovare nella documentazione interna.

Caricamento,controllo e modifica delle regole

Questo modulo prevede il caricamento, il controllo e le modifiche delle regole da un file in un grafo.
Il Caricamento delle regole viene fatto attraverso una funzione la quale legge delle stringhe da un file le quali vengono controllate e validate (secondo un protocollo interno) e inserite in un grafo.
La funzione riempie 2 grafi uno per gli archi di andata e l'altro per glia archi di ritorno tale scelta e stata fatta per il controllo sintattico.
Per il caricamento di stringhe vengono riutilizzate delle funzioni gia utilizzate nei progetti precedenti, mentre il controllo e la validazione viene fatto attraverso una espressione regolare che verra illustrata piu avanti. Ovviamente dopo controllata la stringa verra spezzettata in TAG, creati i nodi e gli archi del grafo e secondo il protocollo interno inseriti nello stesso.
Una volta creato il grafo esso verra analizzato per evitare eventuali ambiguita'.

protocollo interno:

Il protocollo interno prevede la definizione delle regole, dei caratteri speciali e della punteggiatura utilizzata nel file.
REGOLE
Una regola e' una stringa di caratteri seguita dal carattere new_line
Una stringa e' composta da TAG che sono nominati a discrezione dell'utilizzatore e da caratteri speciali che ne indicano un significato semantico.
Un TAG e composto da un testo racchiuso dai caratteri <>
i caratteri speciali utilizzati sono : | & *
il carattere : indica l' inglobamento
il carattere | indica L'operatore booleano OR
il carattere & indica L'operatore booleano AND
il carattere * indica una qualsiasi stringa formata di soli caratteri (non sono accettate stringhe vuote)
ogni stringa deve iniziare con un tag deve essere seguito dai duepunti; dopodiche si puo inserire o un altro tag o il carattere speciale *. Questa e' una striga minima affinche la stringa sia corretta.
Successivamente a questa stringa si possono aggiungere altri tag anteposti dagli operatori booleani.
ATTENZIONE per ogni regola si puo inserire un tipo solo di operatore booleano.
esempio:
<TAG1>:<TAG2>|<TAG3>|* Corretta
<TAG1>:<TAG2>&<TAG3>|* Errata

il carattere speciale * puo essere inserito alla fine della regola dopo il carattere new_line.

Le stringhe su citate vengono validate da una espressione regolare implementata nella funzione della validazione e controllo stringa.

Definizione Semantica

carattere :

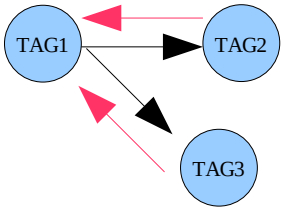
<TAG1>:<TAG2> <TAG1> ingloba <TAG2> significato verbale

significato grafico

<TAG1><TAG2> <\TAG2><\TAG1> significato semantico

carattere |

<TAG1>:<TAG2>|<TAG3> <TAG1> ingloba <TAG2> OR <TAG3> significato verbale

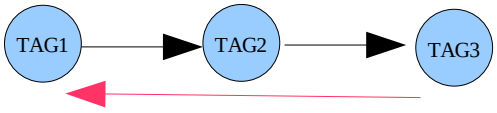


significato grafico

<TAG1>(<TAG2> <\TAG2> oppure <TAG3> <\TAG3>)<\TAG1> significato semantico

carattere &

<TAG1>:<TAG2>&<TAG3> <TAG1> ingloba <TAG2> AND <TAG3> significato verbale



significato grafico

<TAG1><TAG2><\TAG2><TAG3> <\TAG3><\TAG1> significato semantico

Gli archi rossi rappresentano i tag di chiusura nel testo, essi insieme agli archi di apertura formano i possibili cammini affinche un testo secondo queste regole e corretto.

FILE

Il file per ogni regola deve essere concluso con il carattere new_line. Il primo TAG del file indica il TAG di start.

il tag speciale * se messo, deve essere messo alla fine di ogni regola prima del new_line

L'insieme di regole e' ben formato se e solo se esiste un tag di start e un tag di end *

e ogni nodo raggiunga *.

Una regola puo contenere al massimo 256 caratteri.

Dettagli funzionali

Il modulo su citato segue il seguente schema logico:

Caricamento stringa da file
memorizzazione stringa
controllo di validita' stringa (exp. reg.) *(Questa funzione e' utilizzabile solo sotto linux)*
    Se non valida errore stringa
    Se valida
        estrazione dei tag
        inserimento nodi e archi nei grafi (secondo le regole)
        Controllo del grafo

Il caricamento stringa da file e' un pezzo di codice riciclato in progetti addietro implementati. Esso consiste nel caricamento di caratteri da file (fino al new_Line) e inserito in un array statico momentaneo di 256 caratteri.
Memorizzazione stringa prevede l'allocazione di una stringa dinamicamente del numero di caratteri letti in precedenza e memorizzati in essa.
Controllo validita stringa prevede l'implementazione di un funzione che contiene un'espressione regolare che controlla la validita' sintattica della regola.
Espressione regolare :
^(<([[:alnum:]])+>):((<([[:alnum:]])+>\\|)+|(<([[:alnum:]])+>\\&)+)?((<([[:alnum:]])+>)|(*))$
Estrazione dei tag prevede una funzione che scompone la stringa di regole in tag eventualmente indicando l'operatore che li divide.
Inserimento nodi e archi nel grafo prevede una serie di chiamata a funzioni del grafo che inserisco archi e nodi nel grafo seguendo il protocollo interno.
Controllo del grafo prevede una funzione che verifica se nel grafo esiste il nodo di start di end, e in piu controlla se ogni nodo raggiunge il nodo *.
Tale funzione e implementata dal frutto della seguente idea: si applica la BFS sul grafo con archi di andata con nodo sorgente quello di start, si controlla se tutti i nodi sono raggiungibili da esso. Se tale condizione e' vera allora si traspone il grafo e si applica la BFS sul grafo trasposto con nodo sorgente * e si controlla anche questa volta se sono raggiungibili. Se e' vera allora significa che che esistono percorsi che inglobano tutti i nodi che vanno dallo start al nodo *.

La modifica di una regole prevede l'eliminazione di nodi e archi, mentre l'inserimento di una nuova regola prevede l'inserimento di un nuova stringa.

Dettagli implementativi (C oriented)

```
/*Funzione che carica delle regole da file e le inserisce in un grafo( contrlla la validita della regola)
INPUT: path del file di testo delle ragole,Grafo allocato,Array delle conicidenze allocato
OUTPUT: 0 esito positivo altrimeti un intero che indica la prima riga del file d'errore,
grafo delle regole e delle coincidenze riempito
N.B. Ogni regola non deve superare i 256 caratteri*/
int Costruisci_Grafo(char* File_Di_Testo,LIST*** Grafo,void***Coin)

/*Funzione che controlla se la stringa presa in input rispetta le regole del protocollo interno
INPUT:  Stringa da controllare
OUTPUT: 1 se le rispetta,0 altrimenti*/
int Controlla_Regola(char * Stringa)

/*Funzione che inserisce una regola in un grafo
INPUT:  Stringa che contiene la regola,lista di adiacenza delle regole,grafo delle regole con archi di ritorno,grafo con
regole archi di ritorno,array delle coincidenze
OUTPUT: esito inserimento, 1 inserimento corretto, 0 inserimento non corretto
N.b. la reagola deve essere ben fomata secondo il protocollo interno
Grafo e array_coinc sono tripli puntatori nel caso si dovessero fare riallocazioni con cambi di indirizzo*/
int Inserisci_Regola(char *Regola,LIST*** Grafo,LIST*** Grafo_Ritorno,LIST*** Grafo_Ritorno,void***
Array_Coinc)

/*Funzione che cerca una stringa nell'array delle coincidenze, ritorna la posizione
INPUT: Array delle coincidenze,Stringa da cercare
OUTPUT: ritorna l'indice se e' stata trovata altrimenti 0*/
int Search_String(void** Array_Coinc,char* Stringa)

/*Funzione che crea un array di stringhe    <...>    data una regole in input
INPUT: Sringa che contiene la regola,valore di ritorno,valoredi ritorno
OUTPUT: array di puntatori a stringhe allocate,numero di stringhe allocate,Separatore rappresenta l'operatore logico
trovato
es: |  &   oppure 0*/
char** Scompatta_Regola(char* Regola,int *Dim_arr,char *Separatore)

/*Funzione che controlla se il grafo preso in input rispetta le seguenti regole:
Esiste un Nodo di start e che ogni nodo raggiunge il carattere *
INPUT:  Grafo,Array delle coincidenze
OUTPUT: 1 se si verificano le condizioni su cutate ,0 altrimenti*/
int Controlla_Grafo(LIST **Grafo,void** Array_Coinc)
```

Tali funzioni sono implementate nel file gestione_regole.c e i prototipi con le varie definizioni nel file gestione_regole.h

ATTENZIONE IMPORTANTE!!!!

La funzione controllo regola richiama delle funzioni del file header regex.h .Nel quale sono definite le funzioni per la gestione di espressioni regolari. sfortunatamente tali funzioni sono disponibili solo sotto linx, piattaforma sulla quale e stato implementato il progetto.

Sulla piattaforma Windows non sono riuscito a trovare tali file quindi li ho commentati e il controllo delle espressioni regolari non viene fatto. Fate attenzione quando inserire le regole altrimenti potreste avere dei problemi non gestibili.

Controllo Correttezza di un FILE (Secondo le regole)

Tale modulo prevede prevede il controllo sintattico di un file di Testo.
In questo modulo vengono riutilizzate funzioni di progetti addietro.
La funzione principale riutilizzata di tali progetti implementava il seguente schema logico:

Leggi stringa da file
Controlla se la stringa e' un tag di apertura,chiusura,testo
se TAG di Apertura
Semantica di inserimento nello stack

se TAG di Chiusura
Semantica di Estrazione nello stack

se TAG si TESTO
Semantica funzionale di tale input
Se lo stack e' vuoto non ci sono errori

Ovviamente le sezioni che dovranno essere cambiate saranno quelle relative alle semantiche relative alle funzioni sullo stack.

Nel nuovo progetto il vincolo delle regole e' legato ai grafi.
Infatti gli archi di congiunzione dei probabili cammini di correttezza sintattica sono contenuti in due grafi. Un grafo contiene gli arche di andata e un altro gli archi di ritorno.
Gli archi di andata sono quegli archi legati ai TAG di apertura, cioe quelli che mi indicano se e' possibile aprire un nuovo TAG letto da file. Mentre il grafo che contiene gli archi di ritorno sono quegli archi che indicano le possibili scelte quando trovo un TAG di chiusura.
Come si puo trapelare da tale descrizione si capisce che avremo due sensi di marcia(Direzione): andata e ritorno. Se mi trovo nel senso di andata devo controllare nel grafo di andata, se mi trovo nella direzione di ritorno grafo di ritorno.
La direzione cambia quando:
Arrivo ad un testo (cambio in RITORNO)
trovo un tag di apertura (cambio in ANDATA)

Lo scopo di tale funzione e' controllare:
Se i tag aperti vengono chiusi nel giusto ordine
Se i tag rispettano le regole logiche

Quindi sostanzialmente ad ogni tag o testo letto vengono scorsi i grafi e controllati gli eventuali errori.
Fondamentali in questo progetto sono lo stack e il puntatore al nodo attuale.
lo stack mantiene memoria dei tag aperti, il puntatore al nodo attuale mi indica in che punto mi trovo nel grafo.

Semantica degli inserimenti:

Leggo il prossimo tag da file

Se tag di apertura

DIREZIONE di ANDATA

Controllo se il nodo attuale ha come adiacente il nodo letto (nel grafo di andata)

Inserisco sullo stack il nodo letto

Aggiorno il puntatore attuale con il nodo letto

DIREZIONE di RITORNO

Cambio Direzione (andata)

Controllo che l'elemento nello stack abbia come adiacente il tag letto (nel grafo di andata)

Aggiungo allo stack il tag letto

Aggiorno il puntatore attuale con il nodo letto

Se tag di chiusura

DIREZIONE di ANDATA

Estraggo il Nodo dallo stack

Controllo se tale coincide con quello di chiusura

DIREZIONE di RITORNO

Estraggo dallo stack

Controllo se il TAG estratto coincide con il tag letto di chiusura

Controllo se il TAG Attuale ha come adiacente (nel grafo di ritorno) il TaG letto

Aggiorno il TAG attuale con il TAG letto.

Se tag di testo

Controllo che il tag attuale sia *

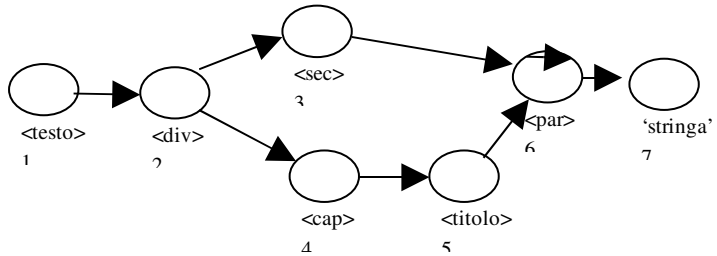
Cambio Direzione (ritorno)

Questo schema e' valido se nello stack c'e' almeno un elemento.
Se lo stack e' vuoto significa che devo inserirgli il tag di start.

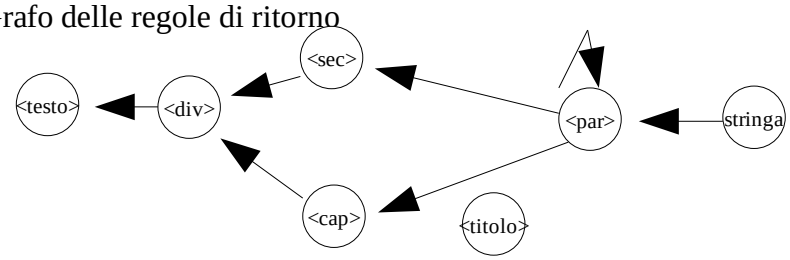
Il controllo sintattico e' corretto se e solo se lo stack e' vuoto.

Esempio:

Grafo delle regole di andata:



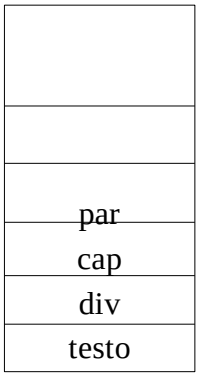
Grafo delle regole di ritorno



Testo

```
<testo>
    <div>
        <cap>
            <titolo>
            <\titolo>
            <par>
                prova
            <\par>
        <\cap>
    <\div>
<\testo>
```

Stack



tag letto=<\par>
Attuale=6
direzione=0 //direzione di ritorno

Dettagli implementativi (C oriented)

```
/*Funzione che controlla la sintassi del file di testo preso in input
INPUT: path del file,array delle coincidenze,Grafo,Grafo delle regole con archi di ritorno
OUTPUT:ritorna Il puntatore alla radice, 0 altrimenti*/
int Controlla_Sintassi(char* File_Di_Testo,void** Array_Coinc,LIST**Grafo,LIST**Grafo_Ritorno)

/*Questa funzione prende un TAG di chiusura e ritorna il nodo di apertura corrispondente
INPUT: Tag di chiusura,Array delle coincidenze
OUTPUT: ritorna 0 se non corrisponde a nessun nodo, altrimenti ritorna l'intero che gli corrisponde
N.B. Il Tag di chiusura e delineato dal carattere \ esempio <\....>*/
int GetTagApertura(char*Tag_Chiusura,void**Array_Coinc)

/*Questa funzione ritorna controlla se la stringa Tag e' di chiusura o apertura
INPUT:stringa Tag
OUTPUT: Ritorna 1 se tag e di apertura, 2 se Tag e' di chiusura, 0 altrimenti  */
IsTag(char * Tag)
```

Tali funzioni sono implementate nel file controllo_sintattico.c metre gli header file nel file controllo_sintattico.h

Calcolo percorso minimo

Questo modulo prevede il calcolo del percorso minimo dal nodo di star al nodo di end *.
In piu prevede l'eliminazione di un nodo o un arco dal grafo e il calcolo del percorso minimo dal nodo di start al nodo di star.
L'idea fondamentale per calcolare il percorso minimo e' quella di eseguire una visita in ampiezza sul grafo (BFS) con nodo sorgente START. Visto che una delle caratteristiche della BFS e quella di ritornare l'albero dei predecessori. Tale albero si dimostra che contiene i percorsi minimi dal nodo sorgente a qualsiasi nodo.
Quindi basta scorrere i predecessori ritornati e si trova il percorso minimo.
La seconda funzione di tale modulo e quella di eliminare un nodo o un arco qualsiasi dal grafo. Tale funzionalita e' gia presente nel menu; quindi basterebbe selezionare la vole di cancellazione nodo o arco e ricalcolare il percorso minimo.
Per semplificare le cose quando viene selezionato questo modulo viene stampato a video il percorso minimo da START a *, viene fatto il trasposto di tale grafo e stampato a video il percorso minimo da * al nodo si start.

Dettagli implementativi (C oriented)

```
/*Questa funzione stampa a video un percorso minimo
INPUT: sorgente,destinazione
OUTPUT: stampa a video*/
Perc_Min(int Sorgente,int Destinazione,LIST** Grafo,void**Array_Coinc);

/*Questa funzione stampa ricorsivamente un un percorso
IPUT: Destinazione,array predecessori,array delle coincidenze
OUTPUT: stampa a video del percorso*/
void Print(int Destinazione ,int*Pred,void**Array_Coinc);
```

Queste funzioni sono implementate nel file percorso_minimo.c e gli header file in percorso_minimo.h

Stampa a video regole

Questo modulo prevede la stampa a video del grafo.
Esso converte la lista di adiacenza in matrice di adiacenza, dopodiche stampa a video l'array delle coincidenze e la matrice di adiacenza.
ovviamente c'e' una relazione 1 a 1 tra l'array e la matrice stessa.
Dove la riga colonna n,m e' uguake a 1 significa che nel grafo c'e' l'arco (n,m)

Dettagli implementativi (C oriented)

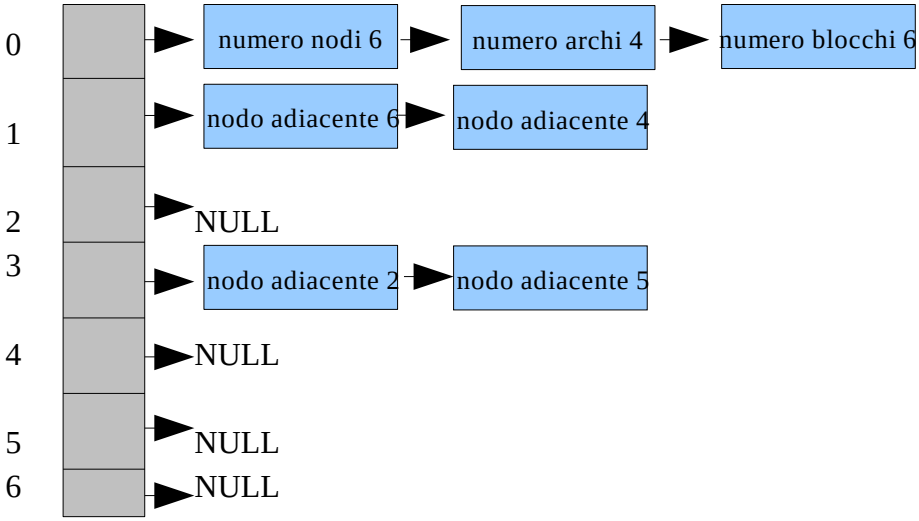
```
/*Funzione che Converte una lista di adiacenza in matrice di adicenza,mettendo ad 1 gli archi
INPUT: Lista di adiacenza
OUTPUT: Matrice di adiacenza
N.B. Il Grafo e' rappresentato come segue: arco(i,j) appartiene al grafo se e solo se Mat_Ad[i][j]=1
la matrice va da (0 a Q_Nodi-1) mentre i nodi nella lista di adiacenza e nella rappresentazione vanno da (1 a Q_Nodi)*/
int **Converti(LIST ** List_Ad)

/*Funzione che stampa a video una grafo rappresentato da una matrice di adiacenza
INPUT: Matrice di adiacenza
OUTPUT: Stampa a video
N.B. Il Grafo e' rappresentato come segue: arco(i,j) appartiene al grafo se e solo se Mat_Ad[i][j]=1
la matrice va da (0 a Q_Nodi-1) mentre i nodi nella lista di adiacenza e nella rappresentazione vanno da (1 a Q_Nodi)*/
void Stampa_Grafo(int ** Mat_Ad,int Q_Nodi)

/*Questa funzione stampa i nomi dell'array di adicenza con i relativi nueri nel grafo
INPUT: Array delle coincidenze
OUTPUT: Stdout
*/
void Stampa_Array_Coinc(void**  Array_Coinc);
```

Tali funzioni sono peculiarita del grafo e sono implementate nel file grafo.c e gli header faile sono nel grafo.h

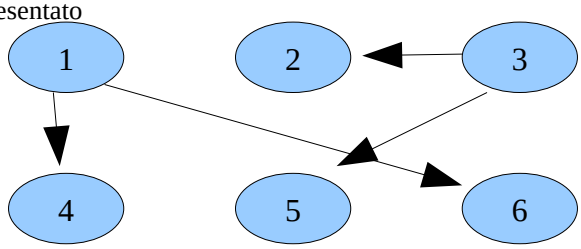
Strutture dati e Routine fondamentali

Le strutture dati fondamentali utilizzate per questo progetto sono i grafi.
I grafi sono delle strutture dati dinamiche , cioe che possono cambiare la loro dimensione a RUN TIME.
I grafi possono essere rappresentati in due differenti modi: tramite le liste di adiacenza oppure tramite matrici di adiacenza.
Le liste di adiacenza rappresentano il grafo come segue:
ogni nodo del grafo possiede una lista di adiacenza, gli elementi contenuti in tale lista sono i nodi adiacenti a tale nodo.
Le matrici di adiacenza rappresentano il grafo come segue:
Esiste una matrice nxn dove n rappresenta il numero di nodi del grafo. Ogni cella riga colonna della matrice rappresenta l'arco di adiacenza di tali nodi.
In questo progetto viene implementata una suit di funzioni per gestire sia le liste che le matrici di adiacenza.
per Quanto riguarda le liste di adiacenza viene riutilizzato la maggior parte del codice sulle liste. Infatti il grafo viene rappresentato come un array di nodi di liste dove l'indice dell' array indica il nodo e al suo interno c'e' un puntatore al primo nodo della lista. I campi Elem (informazione nodo lista) contengono degli interi che rappresentano il nodo di adiacenza.Il primo elemento dell'array contiene una lista composta da 3 nodi i quali contengono informazioni generali relative ai nodi. Nel primo nodo c'e' il numero di nodi del grafo,nel secondo il numero di achi e nel terzo gli spazi di memoria allocati per l'array.
esempio:

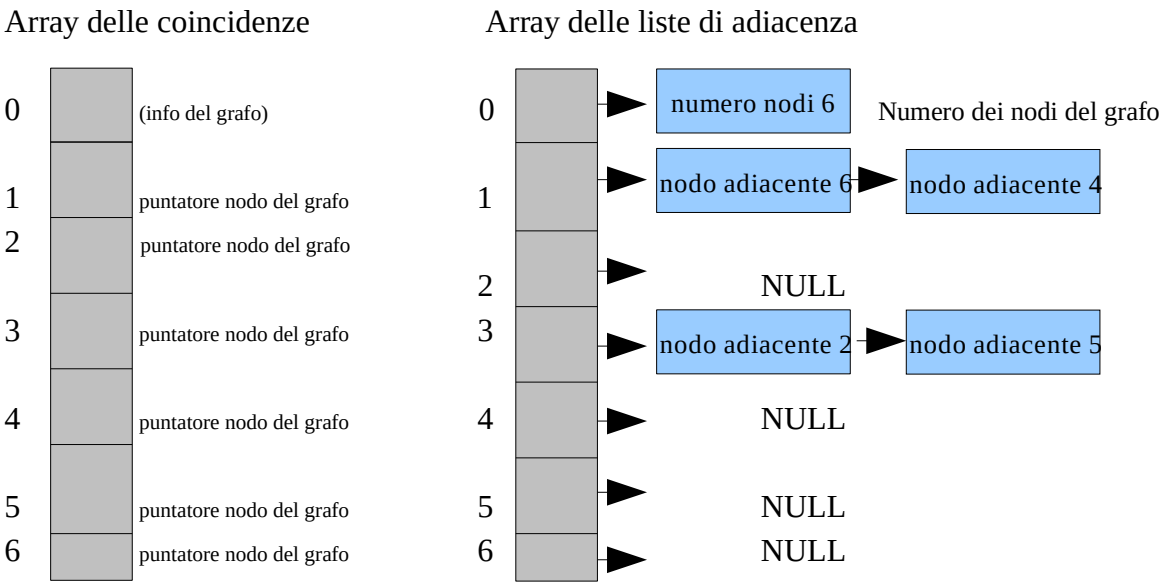
Lista adiacenza



Grafo su rappresentato



A tale array dovra essere associata un altro array della stessa dimensione dell'array su citato, il quale dovra fornire una relazione uno a uno al numero del nodo al nodo del grafo.
Esempio:
Nell'esempio su citato sappiamo che 1 sta in relazione con 4 ma non sappiamo la natura degli elementi del grafo, in questo progetto il numero dovra essere associato ad una stringa. A tale proposito si utilizzera un array di stringhe.



Ogni numero dell'array corrisponde ad un nodo del grafo. Ovviamente il riutilizzo di tale struttura implica la possibilita' di poter assegnare nodi di grafi di diverso genere in progetti differenti.
Tale scoglio potrebbe superarsi utilizzando come array delle coincidenze un array di puntatori void.

Le funzioni primarie applicabili a tale struttera sono:
crea_grafo tale funzione alloca un grafo vuoto con array massimo di 10 elementi.
dealloca_grafo tale funzione dealloca il grafo
inserisci_arco tale funzione inserisce un arco nel grafo
elimina_arco tale funzione elimina un arco nel grafo
inserisci_nodo tale funzione inserisce un nodo nel grafo
elimina_nodo tale funzione elimina un nodo nel grafo
converti_lista tale funzione converte le liste di adiacenza in matrici di adiacenza
stampa_grafo tale funzione stamap il grafo

Per quanto riguarda le matrici di adiacenza tele rappresentazione di grafo viene utilizzata in questo progetto sono per stampare il grafo infatti le funzioni principali vengono fatte sull' altra rappresentazione.
Svantaggi e Vantaggi (delle scelte su fatte):
Scegliere un array di liste e' svantaggioso nel momento in cui bisogna eliminare un nodo; perche se si trova al centro dell'array bisogna ricompattarlo e aggiornare il numero dei nodi; mentre uno dei vantaggi e' legato all'accesso, che e' costante.
Ovviamente anche aggiungere un nodo potrebbe essere svantaggioso perche bisognerebbe ogni

volta riallocare lo spazio dell'array.In questo caso viene allocato dello spazio in piu (in particolare 10 blocchi) che aumenta lo spazio occupato in memoria ma diminuisce il tempo di inserimento.

Un' altra struttura dati dinamica utilizzata sono le code. Esse non sono fondamentali per il progetto in se, ma per un algoritmo di visita sui Grafi (BFS).
Tale struttura dati e' una struttura di dinamica di tipo LIFO cioe' il primo elemento che inserisci e' il primo che togli.
Le funzioni principale utilizzate sono:
Push,Pop,Isempty che servono di conseguenza a inserire un elemento ad estrarre un elemento e controllare se la coda e' vuota.
Visto che tale struttura e' una implementazione particolare di una lista si possono utilizzare alcune funzioni e la struttura dati LIST utilizzata nel file lista.c.
Ricordando sempre che il push si fa in testa (e guarda caso gia e' stata implementata nel file lista.c); e il pop che si fa sempre in coda.

Dettagli implementativi grafo (C oriented)

```
/*Funzione che costruisce e inizializza a NULL un grafo rappresentato da liste di adicenza
INPUT: quantita nodi del grafo da creare
OUTPUT: array di liste, dei nodi creati
N.B. dettaglio implementativo la struttura che contiene i nodi viene allocata di 10 elementi in piu rispetto Q_Nodi*/
LIST **Crea_Grafo(int);

/*Funzione che ritorna il numero di nodi del grafo
INPUT: Lista di adiacenza
OUTPUT: numero di nodi*/
int Get_Num_Nodi(LIST **);

/*Funzione che ritorna il numero di Archi del grafo
INPUT: Lista di adiacenza
OUTPUT: numero di archi*/
int Get_Num_Archi(LIST **);

/*Funzione che ritorna il numero di blocchi dell'array allocati
INPUT: Lista di adiacenza
OUTPUT: numero di blocchi*/
int Get_Num_Blocchi(LIST **);

/*Funzione che dealloca il grafo
INPUT: Lista di adiacenza
ATTENZIONE: dopo la chiamata alla funzione bisogna deallocare anche la variabile al doppio puntatore!!!!
*/
void Dealloca_Grafo(LIST **);
```

```
/*Funzione di deallocazione elemento richiesto dalla funzione su implemetata
INPUT: Elemento da deallocare*/
void Dealloca(void*);

/*Funzione che realloca di 10 nodi il grafo rispetto ai bocchi correnti
INPUT: Lista di adiacenza
OUTPUT: ritorna il grafo reallocato se tutto va a buon fine altrimente NULL*/
LIST ** Realloca_Grafo(LIST **);

/*Funzione che inserisce un arco tra il nodo a e il nodo b
INPUT: Lista di adiacenza,nodo a e nodo b
OUTPUT: Esito dell'inserimeto, 0 errore inseremto, 1 elemento gai inserito,2 se l'inserimento e avvenuto con successo*/
int Ins_Arco(LIST ** ,int , int );

/*Funzione che inserisce un nouvo nodo nel grafo
INPUT: Lista di adiacenza
OUTPUT: 0 errore inseremto, altrimenti l'intero del nuovo nodo
N.B.: controllare se c'e' abbastanza spazio per inserire un nuovo nodo*/
int Ins_Nodo(LIST **);

/*Funzione che elimina un arco A,B nel grafo
INPUT: Lista di adiacenza, Nodo A , Nodo B
OUTPUT: 0 errore Eliminazione,1 Arco inesistente ,2 Eliminazione effettuata con succecco*/
int Elim_Arco(LIST **,int, int);

/*Funzione che elimina un Nodo nel grafo
INPUT: Lista di adiacenza, Nodo da eliminare
OUTPUT: 0 errore Eliminazione,1 Eliminazione effettuata con succecco*/
int Elim_Nodo(LIST **,int);

/*Funzione che elimina una coincidenza
INPUT: Lista di adiacenza, Nodo da eliminare
OUTPUT: 0 errore Eliminazione,1 Eliminazione effettuata con succecco*/
int Elim_Nodo_Coinc(void**,int );

/*Funzione che Converte una lista di adiacenza in matrice di adicenza,mettendo ad 1 gli archi
INPUT: Lista di adiacenza
OUTPUT: Matrice di adiacenza
N.B. Il Grafo e' rappresentato come segue: arco(i,j) appartiene al grafo se e solo se Mat_Ad[i][j]=1
la matrice va da (0 a Q_Nodi-1) mentre i nodi nella lista di adiacenza e nella rappresentazione vanno da (1 a Q_Nodi)*/
int **Converti(LIST **);

/*Funzione che stampa a video una grafo rappresentato da una matrice di adiacenza
INPUT: Matrice di adiacenza
OUTPUT: Stampa a video
N.B. Il Grafo e' rappresentato come segue: arco(i,j) appartiene al grafo se e solo se Mat_Ad[i][j]=1
la matrice va da (0 a Q_Nodi-1) mentre i nodi nella lista di adiacenza e nella rappresentazione vanno da (1 a Q_Nodi)*/
void Stampa_Grafo(int **,int);
```

```
/*Funzione che alloca l'array delle coincidenze
INPUT: Lista di adiacenza
OUTPUT: Array delle coincidenze
N.B. Il primo elemento contiene gli stessi elementi della matrice di adiacenze cioe le info relative al grafo,
     e la dimensione dell'array delle coincidenze ha la stessa dimensione della lista di adiacenza in tutte le sue parti*/
void** Crea_Array_Coinc(LIST **);

/*Funzione che setta l'array delle coincidenze. Associa un numero ad un indirizzo
INPUT: Array delle coincidenze allocato,numero di nodo, Indirizzo da assegnare
OUTPUT: 1 inserimento corretto,0 inserimento non corretto*/
int Set_Coinc(void**,int, void *);

/*Funzione che prende un indirizzo dall'array delle coincidenze.
INPUT: Array delle coincidenze allocato,numero di nodo, Indirizzo da assegnare
OUTPUT: Ritorna l'indirizzo altrimenti NULL*/
void* Get_Coinc(void**,int);

/*Funzione che riadegua il numero di bocchi nel caso fosse stato riallocata la lista di adiacenza
INPUT: Lista delle coincidenze,Lista di adiacenza
OUTPUT: ritorna il nuovo indirizzo se tutto va a buon fine altrimente se non c'e' bisogno di riallocare NULL*/
void** Adegua_Array_Coinc(void**,LIST **);

/*Funzione che ritorna un grafo trasposto
INPUT:Grafo lista di adiacenza
OUTPUT: Grafo trasposto*/
LIST **Trasponi_Grafo(LIST **Grafo);

/*questa funzione implementa una visita in ampiezza di un grafo
INPUT: Grafo , Nodo sorgente da cui iniziare la BFS
OUTPUT:ritorna l'array dei predecessori*/
int* Bfs(LIST**, int );

/*Questa funzione stampa i nomi dell'array di adicenza con i relativi nueri nel grafo
INPUT: Array delle coincidenze
OUTPUT: Stdout */
void Stampa_Array_Coinc(void** );

/*Questa funzione controlla se esite l'arco (A,B) esiste nel grafo rappresentato come liste di adiacenza
INPUT: Grafo,nodo a , nodo b
OUTPUT:ritorna 0 se non esite , altrimenti 1*/
int Esiste(LIST**,int ,int );
```

Tali file sono implementati nel file grafo.c e gli header file nel file grafo.h

Dettagli implementativi grafo (C oriented)

```
/*Questa funzione inserisce un nuovo nodo prima dell'oggetto Nodo, Ritorna la testa.
INPUT se Nodo e' NULL Ritorna il Nodo*/
LIST *Push_Coda(LIST* ,void *);

/*Questa funzione Ritorna la coda della coda.
INPUT: coda
OUTPUT: coda*/
LIST *GetCoda(LIST *);

/*Questa funzione controlla se coda e' vuota.
INPUT: coda
OUTPUT: 0 se non e' vuota 1 se e' vuota*/
int IsEmpty_Coda(LIST *Coda);

/*Questa funzione estrae l'elemento dalla coda, e ritorna in Elem l'indirizzo dell'elemento estratto.
INPUT: coda,puntatore a lista non allocata
OUTPUT: ritnora 0 se la coda e' rimasta vuota, ritorna 1 se la coda e' ancora piena*/
int Pop_Coda(LIST*Coda,LIST**Elem);
```

Tali file sono implementati nel file coda.c e gli header file nel file coda.h

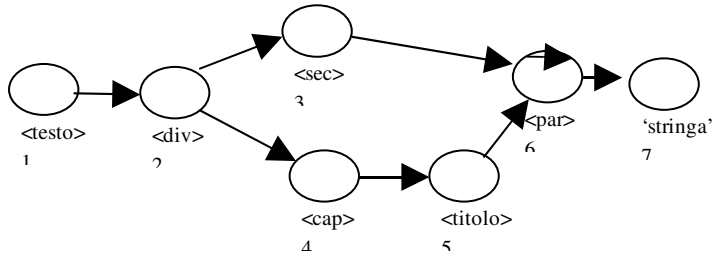
Esempio del progetto

il file REGOLE contiene il seguente contenuto

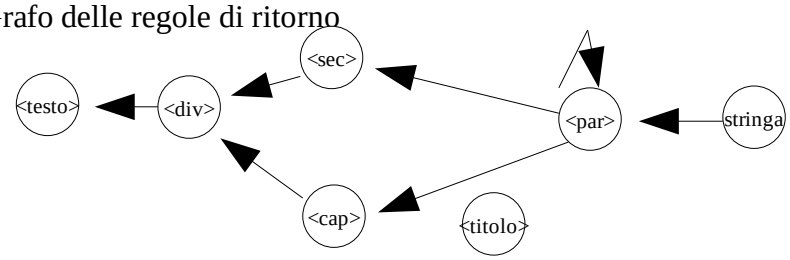
<testo>:
:<sec>|<cap>
<cap>:<titolo>&<par>
<par>:<par>|*
<sec>:<par>

il grafo risultante sara'

Grafo di andata



Grafo delle regole di ritorno



Il file 'testo' contiene un testo ben formato