Bushaj Aldo 847091

Bushaj Antonino 847013

Frisullo Silvestro Stefano 832813

Progetto d'esame parte di programmazione logica

Prolog

Il progetto consiste in un'implementazione in Prolog di alcuni algoritmi per la ricerca applicati al problema "Labirinto".

Il labirinto è una tabella di cui indichiamo il numero di righe e colonne con num_righe(x) e num_colonne(y), con iniziale(pos(x,y))) lo start, finale(pos(x,y)) la o le posizione finali (ossia le uscite) e con occupata(pos(x,y)) i vari ostacoli.

Gli algoritmi utilizzati sono Iterative Deepening, A* e IDA*.

Tutti gli algoritmi sono stati messi alla prova con diversi labirinti in modo da poterne confrontare le prestazioni in diversi casi: labirinto semplice con una uscita, labirinto complesso con una uscita, labirinto senza uscita e labirinto con più uscite.

Tutti e tre i file dell'implementazione degli algoritmi importano il file del labirinto e il file azioni, che ha il compito di decidere la prossima azione e di codificarla in termini di coordinate.

Iterative Deepening algorithm

Questo algoritmo, come suggerisce il nome stesso, lavora in profondità, a differenza dell'esempio trattato a lezione non sarà necessario impostare un valore "limite" alla profondità, ma viene via via incrementata ad ogni chiamata ricorsiva la quale termina solo dopo aver trovato una soluzione, oppure restituisce false nel caso non ce ne siano.

A algorithm*

Questo file utilizza olltre a 'labirinto.pl' e 'azioni.pl' anche 'inversioneLista.pl', 'quick_sort.pl' e 'heuristic.pl' che servono rispettivamente ad invertire una lista, implementare il quick sort per l'ordinamento di una lista di nodi e calcolare l'euristica di un nodo.

IDA algorithm*

I file utilizzati da questo algoritmo oltre i consueti 'azioni.pl', 'applicabile.pl' sono : 'labirinto.pl' e 'heuristic.pl'. In questo algoritmo è stato utilizzato il costrutto findall, il quale scopre tutti i nodi adiacenti a quello su cui ci troviamo, inoltre è stata utilizzata una 'threshold', che rappresenta la soglia massima per l'iterazione corrente .

Confronto e conclusione

Per confrontare le prestazioni di ogni algoritmo sui labirinti è stato usato il predicato time(), che consente di ottenere il tempo di esecuzione in millisecondi, il numero di inferenze e il numero di inferenze per secondo. Nel confronto sono stati utilizzati i primi due parametri perché ritenuti più significativi e meglio rappresentativi delle prestazioni di ciascun algoritmo.

I risultati sono congruenti con le aspettative, per cui iterative deepening è quello che in media ha il maggior numero di inferenze e tempo di esecuzione, mentre quello mediamente più efficiente come numero di inferenze risulta essere A*.

Abbiamo inoltre notato, come è ovvio, una correlazione tra il numero di inferenze, il tempo di esecuzione e la "difficoltà" del labirinto, intesa in termini di grandezza, numero di ostacoli e distanza start-goal.

Le prestazioni in dettaglio sono riportate nel file Statistiche.pdf

Per semplificare la generazione e la traduzione in prolog dei labirinti abbiamo utilizzato uno script python 'mazegen.py', basato sulla libreria mazelib disponibile qui.

Progetto d'esame parte di ASP

La seconda parte del progetto richiedeva la generazione e l'organizzazione di un calendario con le relative ore di lezione con i docenti assegnati con Clingo.
Anche in questo caso abbiamo utilizzato uno script in python che ha il prezioso compito di organizzare l'output di clingo in settimane e giorni, rendendolo molto più leggibile ed intuitivo, quindi basta avviarlo per ottenere un output ben formattato. A seconda del sistema operativo e/o di eventuali shell occorre usare l'istruzione a riga 16 o 17, come spiegato nei commenti al codice. Per eseguire il tutto basta avviare orario.py.

Base di conoscenza

Innanzitutto abbiamo rappresentato i fatti riguardanti le materie e i docenti,
i docenti sono stati espressi tramite i fatti docente(nome), il fatto insegna con due argomenti che indicano il docente e la materia da lui insegnata. Inoltre abbiamo indicato i giorni della settimana tramite dei numeri da 1 a 6 (Lunedì-Sabato), le ore del giorno in totale 8 e le settimane dalla prima alla 24a. l tutto rappresenta la nostra base di conoscenza e quindi i valori su cui andremo successivamente a lavorare.

Vincoli

Sono stati soddisfatti tutti i vincoli richiesti, anche quelli auspicabili. Inoltre a causa di alcune divergenze nell'interpretazione del testo, sono stati aggiunti ulteriori vincoli più stringenti (ma non attivi perchè commentati) sull'esatto numero di ore da svolgere in particolari giorni.

Quando possibile, abbiamo cercato di esplicitare le informazioni come base di conoscenza, ad esempio il fatto che la settimana 7 e 16 si faccia lezione dal lunedì al sabato. L'aggregato #count è stato utilizzato spesso per la necessità di dover contare le ore di lezione.

Abbiamo pensato di assegnare un ID specifico ad ogni ora intesa nella forma (Settimana/Giorno/Ora) in modo tale che nei vincoli fosse possibile riferirsi ad un'ora specifica nell'arco delle 24 settimane, il tutto è stato sviluppato assegnando un peso di '100' alle settimane, '10' ai giorni e '1' alle ore, in questo senso la prima ora della prima settimana ha un valore minore della prima ore della 10a settimana, questo ha semplificato molto la stesura dei vincoli riguardanti la distanza degli inizi dei vari insegnamenti e le propedeuticità.

Conclusione

Nonostante la complessità dell'implementazione, il tempo di esecuzione è relativamente basso, con una media di circa 10 secondi.

In alcuni vincoli abbiamo notato come evitando di fare riferimento ad una variabile quando non necessaria ed utilizzando al suo posto il don't care, si ha un tempo computazionale molto minore.