Esame 20240223

Esercizio 2

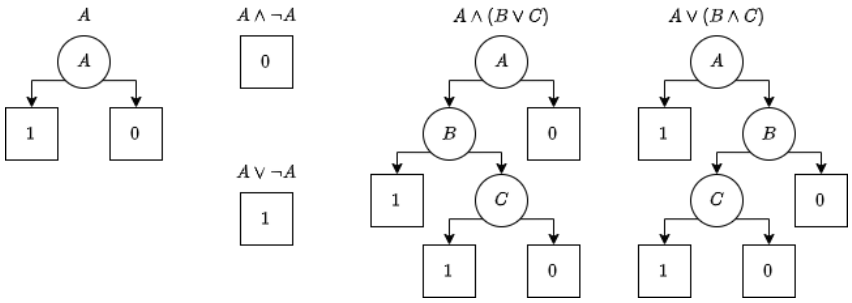
(1) **Esercizio 2 v1**

ESSAY | marked out of 10 | penalty 0 | File picker

In informatica, un albero decisionale ordinato binario è una struttura dati utilizzata per rappresentare una qualunque funzione booleana.

Una qualunque funzione booleana può essere rappresentata come un albero binario costituito da diversi nodi (decisionali) e due nodi terminali. I due nodi terminali sono etichettati 0 (FALSE) e 1 (TRUE). Ciascun nodo (decisionale) u è etichettato da una variabile booleana x_i e ha due nodi figli chiamati figlio di sinistra (o figlio then) e figlio di destra (o figlio else). Seguire il figlio di sinistra (then) rappresenta un'assegnazione del valore TRUE alla variabile x_i, mentre seguire il figlio di destra (else) rappresenta un'assegnazione del valore FALSE alla variabile x_i. Un albero decisionale binario è detto *ordinato* se diverse variabili compaiono nello stesso ordine su tutti i percorsi a partire dalla radice verso le foglie.

Nella figura seguente sono rappresentati diversi alberi decisionali binari ordinat che rappresentano le funzioni booleane A, $A \wedge \neg A$, $A \vee \neg A$, $A \wedge (B \vee C)$, e $A \vee (B \wedge C)$.



Se entrambi i figli di un nodo corrispondente ad una variabile x_i sono nodi isomorfi (ovvero identici), allora la variabile x_i non compare in quella parte dell'albero (si veda l'esempio dell'albero che rappresenta la funzione $A \wedge \neg A$, oppure gli alberi che rappresentano le funzioni $A \wedge (B \vee C)$ e $A \vee (B \wedge C)$).

Dati due alberi decisionali ordinati si può definire l'operazione logica **and** (**congiunzione**) che restituisce un nuovo albero decisionale ordinato che rappresenta la funzione booleana corrispondente all'operazione logica **and** applicata ai due alberi decisionali.

La struttura di questa operazione logica è intrinsecamente ricorsiva, ed è definita come segue:

- **caso base** si controlla se uno dei due alberi è un nodo terminale. Se uno dei due alberi è TRUE si restituisce una copia dell'altro albero, mentre se almeno uno è FALSE si restituisce FALSE.
- **caso ricorsivo** si procede come segue:
  - se le variabili dei due nodi corrispondenti sono uguali, si applica l'operazione logica ai due figli ricorsivamente ottenendo due nuovi alberi T e E, rispettivamente per i casi then ed else. Se T e E sono uguali si restituisce T (o E) e si *distrugge l'altra* (la variabile non influenza il valore di questo ramo della formula), altrimenti si restituisce un nuovo nodo che contiene la variabile corrente e come figli i due nuovi alberi T e E.

– se la variabile del primo albero è **minore** della variabile del secondo albero, si ricorre applicando l'operazione logica al *figlio di sinistra del primo albero* e *al secondo albero*, e al *figlio di destra del primo albero* e *al secondo albero*. Si restituisce un nuovo nodo che contiene *la variabile del primo albero* e come figli i due nuovi alberi ottenuti.
– se la variabile del primo albero è **maggiore** della variabile del secondo albero, si ricorre applicando l'operazione logica al *figlio di sinistra del secondo albero* e *al primo albero*, e al *figlio di destra del secondo albero* e *al primo albero*. Si restituisce un nuovo nodo che contiene *la variabile del secondo albero* e come figli i due nuovi alberi ottenuti.

Nel file `esercizio2.cpp` è presente del codice che definisce una struttura dati `node` che permette di rappresentare un albero decisionale binario ordinato. Il TRUE è rappresentato da un nodo che contiene il carattere '1' e entrambi i figli sono `nullptr`, mentre il FALSE è rappresentato da un nodo che contiene il carattere '0' e entrambi i figli sono `nullptr`. Un nodo generico contiene una lettera dell'alfabeto maiuscola che rappresenta la variabile booleana e due puntatori ai figli. Nel file sono già definite diverse funzioni per manipolare questa struttura dati: `makeNode`, `makeTrue`, `makeFalse`, `deleteNode`, `printNode`, `isTrue`, `isFalse`, `makeCopy`, `makeVar`, `makeNot`, `getVar`, `getThen`, `getElse`, e `areEquivalent`.

Si vuole scrivere un funzione ricorsiva `makeOperation` che implementi secondo lo schema di cui sopra la **congiunzione** (l'**and**) tra due alberi decisionali ordinati. La funzione deve restituire un nuovo albero decisionale ordinato che rappresenta la funzione booleana corrispondente all'operazione logica **and** applicata.

Il file `esercizio2.cpp` contiene già un main con alcuni esempi e alcune invocazioni della funzione `makeOperation`. Di seguito è riportato l'output di esecuzione.

```
marco > a.out
A & B := A(B(1,0),0)
A | B := A(1,B(1,0))
A & (B | C) := A(B(1,C(1,0)),0)
A | (B & C) := A(1,B(C(1,0),0))
res[0] := A(B(1,0),0)
res[1] := 0
res[2] := A(B(C(1,0),0),0)
res[3] := A(B(C(D(1,0),0),0),0)
res[4] := A(B(C(1,0),0),0)
```

Note:

- Scaricare il file `esercizio2.cpp`, modificarlo per inserire la dichiarazione e la definizione della funzione `makeOperation`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`, `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).

esercizio2.cpp

Information for graders:

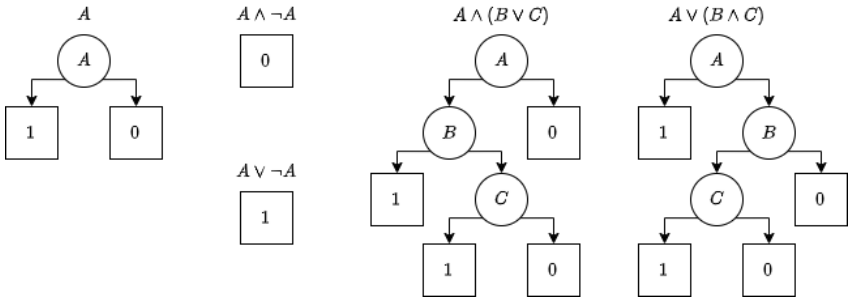
(2) **Esercizio 2 v2**

ESSAY | marked out of 10 | penalty 0 | File picker

In informatica, un albero decisionale ordinato binario è una struttura dati utilizzata per rappresentare una qualunque funzione booleana.

Una qualunque funzione booleana può essere rappresentata come un albero binario costituito da diversi nodi (decisionali) e due nodi terminali. I due nodi terminali sono etichettati 0 (FALSE) e 1 (TRUE). Ciascun nodo (decisionale) u è etichettato da una variabile booleana x_i e ha due nodi figli chiamati figlio di sinistra (o figlio then) e figlio di destra (o figlio else). Seguire il figlio di sinistra (then) rappresenta un'assegnazione del valore TRUE alla variabile x_i, mentre seguire il figlio di destra (else) rappresenta un'assegnazione del valore FALSE alla variabile x_i. Un albero decisionale binario è detto *ordinato* se diverse variabili compaiono nello stesso ordine su tutti i percorsi a partire dalla radice verso le foglie.

Nella figura seguente sono rappresentati diversi alberi decisionali binari ordinat che rappresentano le funzioni booleane A, $A \wedge \neg A$, $A \vee \neg A$, $A \wedge (B \vee C)$, e $A \vee (B \wedge C)$.



Se entrambi i figli di un nodo corrispondente ad una variabile x_i sono nodi isomorfi (ovvero identici), allora la variabile x_i non compare in quella parte dell'albero (si veda l'esempio dell'albero che rappresenta la funzione $A \wedge \neg A$, oppure gli alberi che rappresentano le funzioni $A \wedge (B \vee C)$ e $A \vee (B \wedge C)$).

Dati due alberi decisionali ordinati si può definire l'operazione logica **or** (**disgiunzione**) che restituisce un nuovo albero decisionale ordinato che rappresenta la funzione booleana corrispondente all'operazione logica **or** applicata ai due alberi decisionali.

La struttura di questa operazione logica è intrinsecamente ricorsiva, ed è definita come segue:

- **caso base** si controlla se uno dei due alberi è un nodo terminale. Se uno dei due alberi è FALSE si restituisce una copia dell'altro albero, mentre se almeno uno è TRUE si restituisce TRUE.
- **caso ricorsivo** si procede come segue:
  - se le variabili dei due nodi corrispondenti sono uguali, si applica l'operazione logica ai due figli ricorsivamente ottenendo due nuovi alberi T e E, rispettivamente per i casi then ed else. Se T e E sono uguali si restituisce T (o E) e si *distrugge l'altra* (la variabile non influenza il valore di questo ramo della formula), altrimenti si restituisce un nuovo nodo che contiene la variabile corrente e come figli i due nuovi alberi T e E.
  - se la variabile del primo albero è **minore** della variabile del secondo albero, si ricorre applicando l'operazione logica al *figlio di sinistra del primo albero* e *al secondo albero*, e al *figlio di destra del primo albero* e *al secondo albero*. Si restituisce un nuovo nodo che contiene *la variabile del primo albero* e come figli i due nuovi alberi ottenuti.

– se la variabile del primo albero è **maggiore** della variabile del secondo albero, si ricorre applicando l'operazione logica al *figlio di sinistra del secondo albero* e *al primo albero*, e al *figlio di destra del secondo albero* e *al primo albero*. Si restituisce un nuovo nodo che contiene *la variabile del secondo albero* e come figli i due nuovi alberi ottenuti.

Nel file `esercizio2.cpp` è presente del codice che definisce una struttura dati `node` che permette di rappresentare un albero decisionale binario ordinato. Il TRUE è rappresentato da un nodo che contiene il carattere '1' e entrambi i figli sono `nullptr`, mentre il FALSE è rappresentato da un nodo che contiene il carattere '0' e entrambi i figli sono `nullptr`. Un nodo generico contiene una lettera dell'alfabeto maiuscola che rappresenta la variabile booleana e due puntatori ai figli. Nel file sono già definite diverse funzioni per manipolare questa struttura dati: `makeNode`, `makeTrue`, `makeFalse`, `deleteNode`, `printNode`, `isTrue`, `isFalse`, `makeCopy`, `makeVar`, `makeNot`, `getVar`, `getThen`, `getElse`, e `areEquivalent`.

Si vuole scrivere un funzione ricorsiva `makeOperation` che implementi secondo lo schema di cui sopra la **disgiunzione** (l'**or**) tra due alberi decisionali ordinati. La funzione deve restituire un nuovo albero decisionale ordinato che rappresenta la funzione booleana corrispondente all'operazione logica **or** applicata.

Il file `esercizio2.cpp` contiene già un main con alcuni esempi e alcune invocazioni della funzione `makeOperation`. Di seguito è riportato l'output di esecuzione.

```
marco > ./a.out
A & B := A(B(1,0),0)
A | B := A(1,B(1,0))
A & (B | C) := A(B(1,C(1,0)),0)
A | (B & C) := A(1,B(C(1,0),0))
res[0] := A(1,B(1,0))
res[1] := 1
res[2] := A(1,B(1,C(1,0)))
res[3] := A(1,B(1,C(1,D(1,0))))
res[4] := A(1,B(1,C(1,0)))
```

Note:

- Scaricare il file `esercizio2.cpp`, modificarlo per inserire la dichiarazione e la definizione della funzione `makeOperation`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`, `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).

esercizio2.cpp

Information for graders:

Total of marks: 20