Informatica Generale
Homework 3/2020:
Selection from‘10s Homeworks.

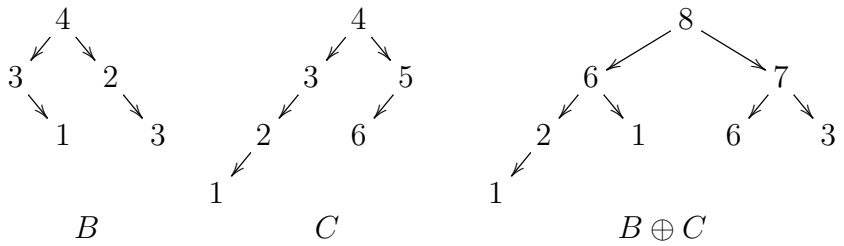
Ivano Salvo – Sapienza Università di Roma – 23/5/2020

Esercizio 1 Scrivere una funzione C di prototipo:

```
binTree sumUnionTree(binTree B, binTree C)
```

che, ricevendo come parametri di input due alberi binari di interi B e C, ritorna come risultato un *nuovo* albero la cui struttura contiene la struttura di entrambi gli alberi e nei nodi comuni contiene come etichetta la somma delle etichette dei nodi corrispondenti negli alberi in input. Nei nodi non comuni, vengono semplicemente ricopiate le etichette presenti nell'albero che la contiene.

Esempio: Dati gli alberi B e C come in figura, l'albero che deve calcolare la funzione è l'albero $B \oplus C$ a destra in figura.



Esercizio 2 (Albero delle chiamate ricorsive, **2018**). Considerate la seguente funzione ricorsiva che calcola i coefficienti binomiali.

```
int cbin(int n, int k){
   if (n==k || n==0) return 1;
   return cbin(n-1,k-1)+cbin(n-1,k);
   }
```

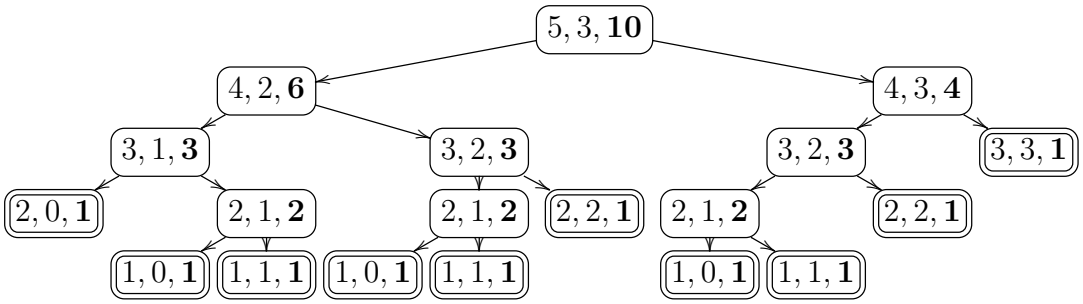


Figura 1: Albero delle chiamate ricorsive di `cbin(5,3)` (in grassetto i risultati ritornati), che deve essere generato dalla funzione `cBinInvocation(5,3)`.

Scrivere una funzione C di prototipo:

```
cBinTree cBinInvocations(int n, int k)
  /* PREC: n>=0, 0<=k<=n */
```

che genera l'albero delle chiamate ricorsive della funzione `int cbin(int n, int k)`, memorizzando in ogni nodo i valori dei parametri n e k e il valore calcolata dalla in tale chiamata. Ogni nodo di un `cbinTree` contiene tre valori (vedi file `cbinTree.h`).

Esempio Se fosse invocata con paramentri $n = 5$ e $k = 3$, la funzione `cBinInvocations` dovrebbe produrre l'albero in Fig. 1, dove il valore ritornato è scritto in grassetto. Notate, in vista dell'esercizio **4**, che ci sono numerosi sotto-alberi ripetuti nell'albero delle chiamate. Osservate anche che, ogni cammino rappresenta le possibili attivazioni di `cbin` che possono essere presenti nello stesso momento sullo stack di sistema.

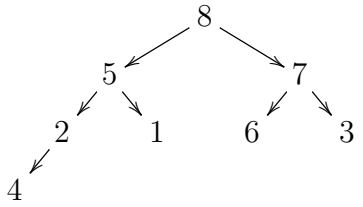
Esercizio 3 (**2017**) Presi due nodi u, v in un albero binario B il *minimo antenato comune* (*low common ancestor*) è il nodo più profondo t, tale che u, v sono entrambi nell'albero radicato in t. Alternativamente, possiamo dire che t è la radice del più piccolo sottoalbero di B che contiene sia u che v. Se u e/o v non fossero presenti in B, il minimo antenato comune sarebbe l'albero vuoto.

Scrivere una funzione C di prototipo:

```
binTree lowCommonAncestor(binTree B, int u, int v)
  /* PREC: B contiene etichette tutte distinte */
```

che restituisce un puntatore alla radice del sottoalbero più piccolo che contiene le etichette u e v. Assumere le etichette dell'albero tutte distinte. Nel caso in cui una o entrambe le etichette non siano presenti, ritornare `NULL`.

Esempio: Nell'albero in figura l'antenato comune di 4 e 2 è l'albero radicato in 2. L'antenato comune tra 4 e 1 è il sottoalbero radicato in 5. L'antenato comune di 3 e 5 è tutto l'albero, cioè l'albero radicato in 8.



Note Pratiche e Concettuali

Potete usare le funzioni contenute nei file `binTree.c` e `cBinTree.c`, che forniscono alcune operazioni base come creazione e stampa di alberi, ma **non dovete consegnarle**. Per compilare il tutto, al solito, dovreste usare i comandi (supponendo le funzioni richieste più eventuali funzioni ausiliarie nei files `LilyEvans.1.c`, `LilyEvans.2.c` e `LilyEvans.3.c`):

```
gcc -std=c99 main-2020-3-1.c binTree.c LilyEvans.1.c
gcc -std=c99 main-2020-3-2.c cBinTree.c LilyEvans.2.c
gcc -std=c99 main-2020-3-3.c binTree.c LilyEvans.3.c
```

Dovete includere le librerie `binTree.h`, `cBinTree.h` (oltre alle librerie standard `stdlib.h`, e `stdio.h`) ovunque necessario. Questo non causa problemi.

Osservate infine che le "nostre" librerie vanno incluse con i comandi:

```
#include "binTree.h"
#include "cBinTree.h"
```

mentre le librerie di sistema con le parentesi angolate. Questo informa il compilatore di cercare nella directory corrente invece che nelle directory in cui sono archiviate le librerie standard.