Prova scritta del corso di:
Algoritmi e Strutture Dati
10 luglio 2023

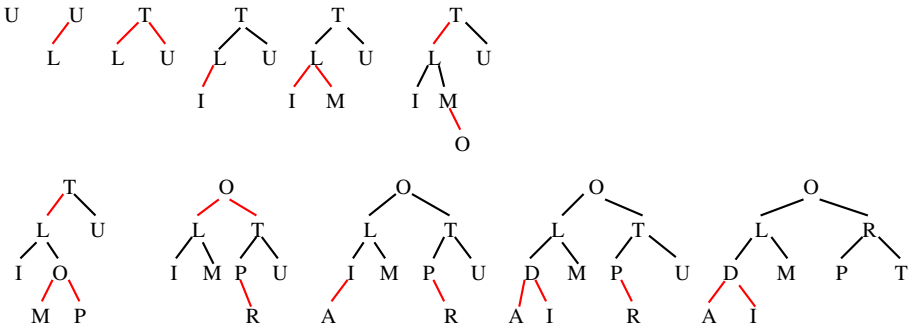
Nome:	.....
Cognome:	.....
Matricola:	.....

1. Date la definizione di albero con radice ordinato e descrivete i diversi modi di rappresentarlo.

Sol. Si vedano i lucidi del corso e il paragrafo 4.6.2 delle dispense. Si noti che possiamo anche utilizzare la rappresentazione tramite liste di adiacenza, con la convenzione che l'ordine con cui sono disposti i nodi in ciascuna lista corrisponda all'ordine definito tra fratelli (primo nodo= primo figlio, secondo nodo=secondo figlio etc. etc.).

2. Considerate la sequenza di inserimenti (a partire dall'albero vuoto) ULTIMOPRAD (se il vostro numero di matricola è pari) o DARPOMITLU (se il vostro numero di matricola è dispari) e disegnate i 10 alberi red-black risultanti al termine di ciascun inserimento. Infine, cancellate la U (se il vostro numero di matricola è pari) o la A (se il vostro numero di matricola è dispari) e disegnate l'albero risultante.

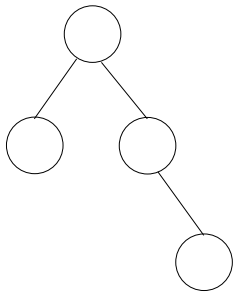
Sol.



3. Quali algoritmi di ordinamento della classe confronti e scambi sono implementabili su lista mantenendo la stessa complessità in tempo della versione su vettore?

Sol. Tutti quelli visti a lezione tranne l'Heapsort. Infatti, quest'ultimo richiede il confronto tra elementi in relazione padre-figlio, confronto che avviene in tempo O(1) tramite vettore (il padre di un elemento di indice i ha indice $i/2$) e $O(n)$ nel caso di lista (occorre scorrere $i/2$ nodi).

4. Dite se eseguendo delle `Union` con bilanciamento (a partire dalla partizione identità) è possibile ottenere un albero dal seguente aspetto. Motivate la risposta.



Sol. L'albero in figura è ottenibile come `Union` di due alberi ciascuno dei quali è formato da due nodi (radice con un solo figlio). Partendo dalla partizione identità $\{\{1\},\{2\},\{3\},\{4\},\{1\}\}$, eseguiamo ad esempio le operazioni `Union`(1,2), `Union`(3,4) e `Union`(1,4).

5. Scrivete una funzione `C(i,j)` che utilizza la programmazione dinamica per calcolare i valori definiti dall'equazione di ricorrenza

$$c(i,j) = 2 \cdot c(i-1,j) + 3 \cdot c(i,j-1)$$

con le condizioni iniziali $c(0,j) = c(i,0) = 1$ per $i,j \geq 0$.

Sol. La funzione utilizza una matrice M di ordine $(i+1) \times (j+1)$ per memorizzare i risultati. Un ordine totale compatibile con quello indotto dall'equazione di ricorrenza è ad esempio

$$c(a,b) < c(m,n) \quad \text{sse} \quad a < m \text{ oppure } a = m \wedge b < n.$$

```
int C(int i, int j){
int M[i+1][j+1]; // alloca spazio per la matrice
int e,m,n;
for(e=0;e<=i;e++) M[e][0]=1; // condizioni iniziali
for(e=1;e<=j;e++) M[0][j]=1;  // condizioni iniziali
for(m=1;m<=i;m++)
 for(n=1;n<=j;n++)
  M[m][n]=2*M[m-1][n]+3*M[m][n-1];
return M[i][j];
}
```

Il programma ha complessità $\Theta(i \cdot j)$ sia in tempo, sia in spazio (secondo il criterio di costo uniforme).