algoritmi e strutture di dati

esercizi di ricapitolazione

m.patrignani

esercizi sulle liste (1)

- scrivi lo pseudocodice della procedura `TOGLI_MINORI`(*l*, *k*) che elimina dalla lista di interi doppiamente concatenata *l* tutti gli elementi di valore minore di *k*, inserendoli in un'altra lista (in qualunque ordine) che viene restituita in output

- scrivi lo pseudocodice della procedura `MULTIINSIEME`(*l*) che ritorna `TRUE` se una lista di interi doppiamente concatenata *l* rappresenta un multiinsieme (cioè un insieme con elementi ripetuti), altrimenti, se tutti gli elementi sono distinti, ritorna `FALSE`

esercizi sulle liste (2)

- scrivi lo pseudocodice della procedura `INTERSEZIONE`(l_1,l_2) che accetti come input due insiemi di interi, rappresentati tramite liste doppiamente concatenate, e restituisca in output una lista doppiamente concatenata che rappresenti l'intersezione dei due insiemi

- scrivi lo pseudocodice della procedura `UNIONE`(l_1,l_2) che accetti come input due insiemi di interi, rappresentati tramite liste doppiamente concatenate, e restituisca in output una lista doppiamente concatenata che rappresenti l'unione dei due insiemi

esercizi sugli alberi binari (1)

- scrivi lo pseudocodice della funzione `CONTA_MINORI`(t) che accetti in input un albero binario t e restituisca il numero di nodi di t che hanno un valore della chiave minore di quello del genitore
  - la radice, non avendo genitore, non deve essere contata

esercizi sugli alberi binari (2)

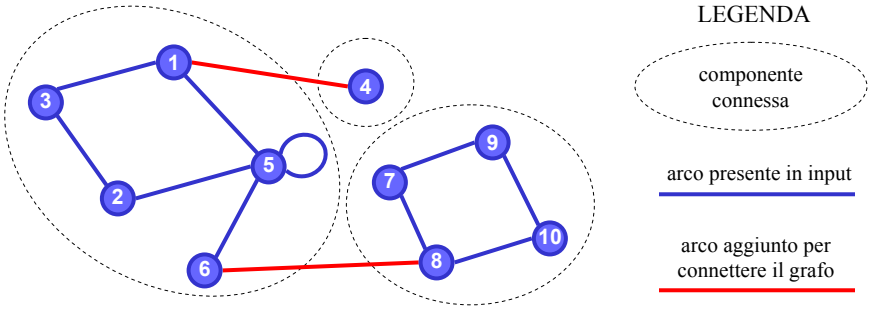
- scrivi lo pseudocodice della funzione `CONTA_MAGGIORI`(t) che accetti in input un albero binario t e restituisca il numero di nodi di t che hanno un valore della chiave maggiore della somma dei valori dei loro discendenti

alberi di grado arbitrario

- scrivi lo pseudocodice la funzione `MERGE`(t_1,t_2) che accetti in input due alberi di grado arbitrario t_1 e t_2, in cui si assume che il campo info della radice di t2 sia uguale a quello di una foglia di t1, e produca in output un albero t_3 che abbia gli stessi nodi ed archi di t_1 e di t_2
  - gli alberi t_1 e t_2 devono rimanere invariati
  - gli alberi sono rappresentati con la struttura figlio-sinistro fratello-destro

visite di grafi

- scrivi lo pseudocodice la funzione `CONNETTI`(A) che accetti in input un grafo non orientato descritto tramite un array A di liste di adiacenza doppiamente concatenate, calcoli le componenti connesse del grafo e modifichi il grafo stesso aggiungendo gli archi che sono sufficienti per produrre un grafo connesso



grafi e alberi (1)

- scrivi lo pseudocodice la funzione `GRAFO-DA-ALBERO`(t) che accetti in input un albero di grado arbitrario t (con i nodi numerati da 0 ad n-1) e produca in output un grafo non orientato A con gli stessi archi e nodi dell'albero t
  - l'albero t è rappresentato con la struttura figlio-sinistro fratello-destro
  - il grafo A è rappresentato da un array di riferimenti al primo elemento delle liste di adiacenza (doppiamente concatenate)

grafi e alberi (2)

- scrivi lo pseudocodice la funzione `SEMPLIFICA`(A,t) che accetti in input un grafo non orientato A e un albero ricoprente di A di grado arbitrario t e rimuova dal grafo A tutti gli archi che fanno parte dell'albero t
  - il grafo A è rappresentato da un array di riferimenti al primo elemento delle liste di adiacenza (doppiamente concatenate)
  - l'albero t è rappresentato con la struttura figlio-sinistro fratello-destro

grafi e alberi (3)

- scrivi lo pseudocodice la funzione `SEMPLIFICA`(A,t) che accetti in input un grafo non orientato A e un albero ricoprente di A di grado arbitrario t e rimuova dal grafo A tutti gli archi che non fanno parte dell'albero t
  - il grafo A è rappresentato da un array di riferimenti al primo elemento delle liste di adiacenza (doppiamente concatenate)
  - l'albero t è rappresentato con la struttura figlio-sinistro fratello-destro

soluzioni

soluzioni: TOGLI_MINORI

```
TOGLI_MINORI(l, k)
    /* lout è una nuova lista doppiamente concatenata di interi */
    lout.head = NULL   /* inizializzazione (non indispensabile) */
    x = l.head
    while x != NULL
            temp = x.next       /* potrei dover rimuovere x, perdendo x.next */
            if (x.info < k)
                        AGGIUNGI(lout,x.info)
                        RIMUOVI(l,x)
            x = temp
    return lout
```

```
AGGIUNGI(l,v)
    /* temp è un nuovo elemento della lista */
    temp.info = v
    temp.prev = NULL
    temp.next = l.head
    if (l.head != NULL)
            l.head.prev = temp
    l.head  = temp
```

```
RIMUOVI(l,x)  /* suppongo x appartenga ad l */
    if x.prev != NULL /* non è primo elemento */
            x.prev.next = x.next
    else
            l.head = x.next
    if x.next != NULL /* esiste successivo */
            x.next.prev = x.prev
```


soluzioni: MULTIINSIEME

```
MULTIINSIEME(l)
    x = l.head
    while x != NULL
        y = x.next    /* confronto x con i successivi */
        while y != NULL
            if x.info == y.info
                return TRUE /* trovato doppione */
            y = y.next
        x = x.next
    return FALSE
```


soluzioni: INTERSEZIONE

```
INTERSEZIONE(l1,l2)
    /* lout è una nuova lista doppiamente concatenata di interi */
    lout.head = 0                /* inizializzazione (non indispensabile) */
    x = l1.head
    while x != NULL
        if TROVATO(l2, x.info)
            AGGIUNGI(lout, x.info)
        x = x.next
    return lout

TROVATO(l,v)
    x = l.head
    while x != NULL
        if (x.info == v) return TRUE
        x = x.next
    return FALSE
```

Nota: per la funzione AGGIUNGI vedi le soluzioni dei precedenti esercizi

soluzioni: UNIONE

```
UNIONE(l1,l2)
    /* lout è una nuova lista doppiamente concatenata di interi */
    lout.head = 0   /* inizializzazione (non indispensabile) */
    x = l1.head
    while x != NULL
            AGGIUNGI(lout, x.info)
            x = x.next
    x = l2.head
    while x != NULL
            if !TROVATO(lout, x.info)
                    AGGIUNGI(lout.x.info)
            x = x.next
    return lout
```

Nota: per le funzioni AGGIUNGI e TROVATO vedi le soluzioni degli esercizi precedenti

soluzioni: CONTA_MINORI

```
CONTA_MINORI(t)
    return CONTA_RIC(t.root)

CONTA_RIC(n)
    minori = 0
    if (n == NULL) return minori
    if (n.parent != NULL)
            if (n.info < n.parent.info)
                    minori = 1
    return minori + CONTA_RIC(n.left) + CONTA_RIC(n.right)
```

Nota: la complessità della funzione è $\Theta(n)$, dove n è il numero di nodi dell'albero t

soluzioni: CONNETTI (1)

```
CONNETTI(A)
    cont = 0   /* contatore delle componenti connesse */
    /* color è un array di interi con A.length posizioni */
    for i = 0 to color.length-1
            color[i] = 0
    for i = 0 to A.length-1
            if (color[i]==0)
                    cont = cont + 1        /* nuova componente connessa */
                    DFS(A, i, color, cont)          /* visito e marco con cont */
    /* rapp è un array di interi di dimensione cont+1. rapp[i] è l'indice di un nodo
    qualsiasi che appartiene alla componente connessa i-esima (rapp[0] non è utilizzato)
    */
    for i = 1 to cont
            for j = 0 to color.length-1
                    if (color[j] == i)
                            rapp[i] = j
                            j = color.length /* per interrompere la scansione (break) */
    for i = 2 to cont
            AGGIUNGI_ARCO(A,rapp[1],rapp[i])
            AGGIUNGI_ARCO(A,rapp[i],rapp[1])
```


soluzioni: CONNETTI (2)

```
DFS(A, i, color,k)  /* visito e marco tutto con k */
    color[i] = k        /* visitato*/
    x = A[i]
    while x != NULL
            v = x.info
            if (color[v] == 0)
                    DFS(A, v, color, k)
            x = x.next


AGGIUNGI_ARCO(A,i,j)
    /* temp nuovo elemento della lista */
    temp.info = j
    temp.prev = NULL
    temp.next = A[i]
    if (A[i] != NULL)
            A[i].prev = temp
    A[i] = temp
```


soluzione alternativa: CONNETTI

```
CONNETTI(A)
   /* color è un array di interi con A.length posizioni */
   for i = 0 to color.length-1
         color[i] = 0
   for i = 0 to A.length-1
         if (color[i]==0)
               DFS(A, i, color, cont) /* visito e marco con cont */
               if (i != 0)
                     AGGIUNGI_ARCO(A,0,i)
```

