Corso di Laurea in Informatica – Algoritmi e Strutture Dati – Corso M-Z – 2024/2025
IV Appello – 09/06/2025, 10.00-12.00

Consegnare un unico file PDF chiamato COGNOME_NOME.pdf

----------Es. 1 Estensione di una struttura dati esistente, con nuovi operatori (5 punti)

Data la struttura dati **albero n-ario**, scrivere specifica sintattica e semantica di due nuovi operatori (quando possibile/utile, fare uso degli operatori esistenti dell'albero):
- **figli_ordinati,** che prende un nodo *u* e restituisce la lista dei nodi figli di *u*, ordinati in senso crescente per valore;

figli_ordinati(Nodo, Albero)-> Lista<Nodo>
figli_ordinati(u, T)-> L
PRE: T = <N, A>, u ∈ N
POST: se foglia(u,T), allora L = <>
altrimenti L = <a1, a2, …, an>, per ogni i, 1<=i<=n (ai ∈ N AND PADRE(ai,T) = u)
AND per ogni j, 1<=j<n, leggiNodo(aj,T)<=leggiNodo(a(j+1),T) // impone ordinamento
AND per ogni i, 1<=i<=n, per ogni j, 1<=j<=n, con i!=j, ai != aj // evito duplicati nella lista
AND dato m = |{x ∈ N | padre(x,T) = u}|, n = m // garantisco di prendere TUTTI i figli
OPPURE
AND per ogni x ∈ N | padre(x,T) = u, esiste i, 1<=i<=n, t.c. x = ai // garantisco di prendere TUTTI i figli

- **pari,** che prende un nodo *u* e restituisce *true*, se il valore contenuto è pari; *false* altrimenti.

pari(Nodo, Albero) -> bool
pari(u, T) -> b
PRE: T = <N, A>, u ∈ N
POST: b = true se leggiNodo(u,T) % 2 = 0; b = false altrimenti.

Es. 2. Descrivere le implementazioni del **grafo** basate su **matrice di adiacenza** e **matrice di adiacenza estesa**. Discutere (enunciare e motivare) la complessità computazionale di ogni singolo operatore del grafo implementato attraverso tali approcci, evidenziandone differenze e vantaggi/svantaggi. **(8 punti)**

Es. 3. Si vuole progettare una struttura dati per la memorizzazione di un social network dedicato a ricercatori scientifici. Ogni ricercatore è eventualmente associato ad altri ricercatori, nel caso in cui abbiano collaborato alla scrittura di articoli scientifici. È necessario memorizzare il numero di articoli scientifici per cui hanno collaborato. Per ogni ricercatore, è inoltre necessario memorizzare i riconoscimenti ottenuti a livello internazionale. Completare la specifica della struttura dati **social**, fornendo la specifica semantica per mezzo di PRE e POST condizioni, rispetto alla seguente specifica sintattica:
domini: social, ricercatore, riconoscimento (aggiungerne altri se necessario)
social: insieme dei grafi non orientati G = <N, A>, con nodi di tipo *ricercatore* e valore dei nodi di tipo *insieme<riconoscimento>* e valore degli archi di tipo *intero*
operatori:

// crea il social network, inizialmente senza alcun ricercatore (1p)
creaSocial () → social

creaSocial() -> s
PRE: -
POST: s = <{}, {}>

// aggiunge un ricercatore al social network (1p)
aggiungiRicercatore (social, ricercatore) → social

aggiungeRicercatore(s, r) -> s'
PRE: s = <N, A>, !esisteNodo(r, s)
POST: s' = scriviNodo(r, insNodo(r,s), {})

// memorizza il fatto che due ricercatori hanno collaborato alla scrittura di un nuovo articolo scientifico (2p)
aggiungiCoautori (social, ricercatore, ricercatore) -> social

aggiungiCoautori(s, r1, r2) -> s'
PRE: s = <N, A>, esisteNodo(r1, s), esisteNodo(r2, s)
POST:
SE esisteArco(r1, r2, s), s' = scriviArco(r1, r2, s, leggiArco(r1,r2,s)+1)
ALTRIMENTI s' = scriviArco(r1, r2, insArco(r1,r2,s), 1)
OPPURE
dato x = insArco(r1,r2,s), s' = scriviArco(r1, r2, x, 1)

// aggiungi un riconoscimento ricevuto da un ricercatore (2p)
aggiungiRiconoscimento (social, ricercatore, riconoscimento) -> social

aggiungiRiconoscimento(s,r,p) -> s'
PRE: s = <N, A>, esisteNodo(r, s)
POST: s' = scriviNodo(s,r, leggiNodo(s,r) U {p})

// restituisce il ricercatore che ha scritto il maggior numero di articoli con altri ricercatori; in caso di parimerito, restituire tutti quelli a parimerito (3p)
piuArticoli(social) -> **??** (determinare il tipo dell'output) **(9 punti)**

piuArticoli(social)->insieme<ricercatore>
piuArticoli(s)->I
PRE: s = <N, A>
POST: I = { r ∈ N | non esiste y ∈ N t.c. contaArticoli(s,y) > contaArticoli(s,r)}

contaArticoli(social, ricercatore)->intero
contaArticoli(s,r)->i
PRE: s = <N, A>, r ∈ N
POST: i = sommatoria per ogni [x ∈ N AND esisteArco(r,x,s)] di leggiArco(r,x,s)

Es. 4. Tecniche algoritmiche **(7 punti)**
Spiegare il problema dell'allocazione di attività ad una risorsa condivisa e proporre due soluzioni algoritmiche, una basata su tecnica enumerativa e una basata su tecnica greedy, confrontandone la complessità computazionale.

Es. 5. Si consideri il seguente algoritmo che lavora sugli array A e B, aventi numero di elementi pari a *n*. Stimare la complessità dell'algoritmo in funzione di n nel caso pessimo e nel caso ottimo, motivando la risposta e illustrando in quali casi l'algoritmo si trova nelle condizioni ottime e pessime.

(4 punti)

```
void analizza_array(int A[], int B[], int n)
{
  int conta = 0, soglia = 5, i = 0;
  while(i<n && conta <= soglia)
  {
     if(A[i] == B[i])
        conta++;
     i++;
  }
  if(conta >= soglia)
     selection_sort(A);
}
```

Laboratorio

La prova di laboratorio non sarà corretta se non si supera la prova scritta.

Si assuma di avere una classe C++ per l'implementazione di alberi n-ari in sola lettura, che presenti le seguenti funzioni:

```
template< class T >
class ReadOnlyTree {
 public:
        typedef int Nodo;

        ReadOnlyTree();
        bool vuoto() const;
Nodo radice() const;
        Nodo padre(Nodo) const;
        Nodo primofiglio(Nodo) const;
        Nodo succfratello(Nodo) const;
        bool ultimofratello(Nodo) const;
        bool foglia(Nodo) const;
leggi(Nodo) const;
void scrivi(Nodo, const T &) const;
}
```

Scrivere la funzione *confronta_alberi (...)* che prenda in input due alberi n-ari di interi *P* e *Q*, e un valore intero *k*, e restituisca una lista contenente i nodi di *P* il cui valore è presente in almeno k nodi dell'albero *Q*.