Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

Prova di laboratorio - Appello del 9 luglio 2020

Esercizio 1

Le funzioni qui riportate usano due algoritmi differenti per calcolare entrambe la potenza x^n tra due interi x e n. Date una prima lettura, senza rispondere, alle domande sotto, quindi familiarizzate con il codice; solo dopo aver fatto ciò, rispondete alle domande, giustificando le vostre risposte.

```
1  int pow1( int x, int n) {
2    if (n == 0)
3      return 1;
4    if (n == 1)
5      return x;
6    if (n%2 == 1)
7      return pow1(x*x,n/2) * x;
8    return pow1(x*x,n/2);
9  }
```

```
1  int pow2( int x, int n) {
2    int res = 1;
3    int i = 1;
4    while ( i<=n ) {
5      res *= x;
6      i++;
7    }
8    return res;
9  }
```

Per ciascuna funzione, rispondete alle seguenti domande:

1. Descrivete con parole vostre come funziona l'algoritmo (senza usare esempi). Cercate di spiegare l'algoritmo in generale, non riga per riga.
2. In quale riga viene eseguita la *prima* moltiplicazione? Tra quali fattori?
3. In quale riga viene eseguita l'*ultima* moltiplicazione? Tra quali fattori?
4. Quante moltiplicazioni sono eseguite in tutto dalla funzione e in quali righe?
5. Tracciate l'esecuzione dell'algoritmo per $x = 2$ e $n = 9$.
6. Analizzate la complessità in tempo dell'algoritmo in funzione di n.

Esercizio 2

In questo esercizio consideriamo la tipica implementazione di una lista concatenata, dove il tipo `List` è definito come segue (se `head` è `NULL`, allora la lista è vuota):

```
1  struct node {
2    int item;
3    struct node *next;
4  };
5
6  typedef struct {
7    struct node *head;
8  } *List;
```

1. Scrivete una funzione `void list_print(List l)` che stampa la lista puntata da `l`.
2. Scrivete una funzione `void list_addAtEnd(List l, int value)` che inserisce in coda alla lista `l` un nodo con valore `value` (potete assumere che `l` sia diverso da `NULL`).
3. Definite un nuovo tipo `DList` con due membri `head` e `tail` per realizzare una lista doppiamente concatenata; quindi scrivete una funzione `void dlist_addAtEnd(DList *l, int value)` che inserisce in coda alla lista doppiamente concatenata `l` un nodo con valore `value` (anche qui potete assumere che `l` sia diverso da `NULL`).
4. Confrontate la complessità in tempo delle due funzioni `list_addAtEnd` e `dlist_addAtEnd`.
5. Riassumete con parole vostre quale è lo scopo della funzione `mystery` qui sotto riportata, prendendo in considerazione tutti i casi possibili.

```
1  struct node *mystery( List l, int value ) {
2    struct node *current = l -> head;
3    struct node *temp = NULL;
4    while ( current != NULL && current -> item != value ) {
5      temp = current;
6      current = current -> next;
7    }
8    return temp;
9  }
```