Corso di Informatica@Matematica
Prova scritta del 10 febbraio 2016

Tempo a disposizione: ore 2.
Esercizi (1, 2, 3) e (4, 5) su due fogli differenti. Es. 3 e 5 necessari per la sufficienza.
In entrambi gli esercizi è possibile definire funzioni ausiliarie.

1. (6 punti) Il codice ASCII su 7 bit della cifra n corrisponde alla codifica binaria (assoluta) di $48+n$. Si supponga di estendere tale codifica con uno "1" più significativo e di disporre da sinistra (più significativo) a destra le codifiche di 6 e 3 ottenute nel modo descritto. Si interpreti ora tale sequenza di sedici bit come un numero in complemento a due e lo si cambi di segno. Si dia la rappresentazione in ottale del risultato così ottenuto.

   Sol: La codifica ASCII (sette bit) di 6 è `011 0110`; quella di 3 è `011 0011`. Estendendo entrambe a sinistra con 1 e giustapponendole si ottiene `1011 0110 1011 0011`. Cambiamo il segno, lavorando in complemento a due: `0100 1001 0100 1101`. Raggruppiamo a tre a tre da destra e convertiamo ogni tripletta in ottale (separatamente): `0 100 100 101 001 101`, da cui `044515`, che è quanto richiesto.

2. (6 punti) Cosa stampa il seguente frammento Python?

```
T=(1,2)
L=T
print(T is L)
T=T+(3,)
print(T)
print(L)
L = L+(4,)
print(T is L)
print(T)
print(L)
```

   Sol: `True, (1, 2, 3), (1, 2), False, (1, 2, 3), (1, 2, 4)`

3. (7 punti) Si scriva una funzione Python `sdoppia(A)` che, presa una sequenza `A`, restituisce una tupla composta dagli stessi elementi di `A` e nello stesso ordine, ma nella quale eventuali elementi *contigui* ripetuti (in `A`) compaiono una sola volta. Esempio: `[1,2,2,-1,0,0,2,2,2,2,3]` deve restituire `(1,2,-1,0,2,3)`.

   Sol: E' sufficiente una semplice scansione lineare, con gestione a parte del caso vuoto e dell'inizializzazione:

```
def sdoppia(A):
  if len(A)==0:
    return ()
  R=(A[0],)
  for i in range(1,len(A)):
     if A[i]!=R[-1]:
        R = R+(A[i],)
  return R
```

4. (6 punti) Cosa stampa il seguente frammento di codice Python (si possono usare artifici grafici per descrivere l'output):

```
a=[1,2,3]
b=['a','b','c']
c=['X','Y','Z']

def f(a):
    a[1]='X'
    b[2]=0
    return c

a=f(b)
c = c.append(b)
print(a,b,c)
```

   Sol: `['X', 'Y', 'Z', ['a', 'X', 0]] ['a', 'X', 0] None`

5. (7 punti) Si supponga di avere a disposizione le seguenti funzioni Python, che permettono di costruire e manipolare alberi binari (come sono realizzate non è significativo per l'esercizio):

```
def tree():             # Restituisce l'albero vuoto
def bin(a,l,r):         # Costruisce (restituisce) l'albero con radice etichettata a,
                        # figlio sinistro l e figlio destro r
def is_empty(a):        # Restituisce True sse a e' l'albero vuoto
def label(a):           # Restituisce l'etichetta della radice dell'albero a
def first_child(a):     # Restituisce il figlio sinistro dell'albero a
def second_child(a):    # Restituisce il figlio destro dell'albero a
```

Si consideri un albero binario `a` etichettato con valori di tipo `float`. Scrivere una funzione `contanodi(a, x, y)` che, dato in input l'albero `a` e due valori `x` e `y`, con `x < y`, restituisce il numero di nodi etichettati con un valore compreso nell'intervallo `[x, y]` (estremi inclusi). Gli alberi possono essere manipolati *solo* con le funzioni definite sopra.

Sol: La soluzione modulare suddivide il problema in due passi:

- estrazione della lista dei nodi dell'albero (in un ordine qualsiasi)
- conteggio dei valori compresi nell'intervallo presenti nella lista

```
def nodi(a):
    if is_empty(a):
        return [];
    r = [label(a)]
    return r + nodi(first_child(a)) + nodi(second_child(a))

def conta_valori(l, x, y):
    n = 0
    for i in l:
        if i >= x and i <= y:
            n = n + 1
    return n

def contanodi(a, x, y):
    l = nodi(a)
    return conta_valori(l, x, y)
```

Possiamo dare anche un'altra soluzione ricorsiva diretta, che distingue due casi:

- *albero vuoto*: il risultato è `0`
- *albero con almeno un nodo*: si esegue `contanodi` prima sul sottalbero destro, poi sul sottoalbero sinistro e quindi si sommano i due valori ottenuti. Al risultato si aggiunge uno se l'etichetta della radice è compresa nell'intervallo, 0 altrimenti.

```
def contanodi(a, x, y):
    if is_empty(a):
        return 0
    fc = contanodi(first_child(a, x, y))
    sc = contanodi(second_child(a, x, y))
    r = label(a)
    if r >= x and r <= y:
        return 1 + fc + sc
    else:
        return fc + sc
```