ALGORITMICA Verifica del 16 Maggio 2002

COGNOME **NOME** **CORSO**

Esercizio 1. *(10 punti)* Si vuole inserire la sequenza di chiavi intere 19, 36, 6, 132, 39, 262, 45, 67, 64, 28, 15, 60 in una tabella hash inizialmente vuota di lunghezza $m = 16$, usando l'indirizzamento aperto con scansione quadratica mediante la seguente funzione hash per ciascuna chiave k:

$$h(k,i) = \left(h'(k) + \frac{1}{2}i + \frac{1}{2}i^2\right) \bmod m, \quad \text{dove } i = 0, 1, \ldots, \text{ e } h'(k) = k \bmod m.$$

Mostrare l'esecuzione delle inserzioni elencando le posizioni $h(k,i)$ esaminate nella tabella hash durante la scansione operata con ciascuna chiave k.

chiave k	Posizioni $h(k,i)$ esaminate nella tabella hash
19	
36	
6	
132	
39	
262	
45	
67	
64	
28	
15	
60	

Soluzione:

chiave k	Posizioni $h(k, i)$ esaminate nella tabella hash
19	3
36	4
6	6
132	4, 5
39	7
262	6, 7, 9
45	13
67	3, 4, 6, 9, 13, 2
64	0
28	12
15	15
60	12, 13, 15, 2, 6, 11

ALGORITMICA Verifica del 16 Maggio 2002

COGNOME **NOME** **CORSO**

Esercizio 2. *(10 punti)* È dato un grafo non diretto $G = (V, E)$ rappresentato con liste di adiacenza, in cui ogni vertice $u \in V$ ha nome NOME(u), ove NOME è un array di $|V|$ elementi. Dato un vertice $x \in V$ e un intero k ($1 \leq k \leq n-1$), si deve stabilire se esiste un'altro vertice $y \in V$, tale che NOME(y) = NOME(x) e il cammino minimo tra x e y contiene almeno k archi.

1. Descrivere a parole la struttura dell'algoritmo di risoluzione.
2. Dare una realizzazione dell'algoritmo in pseudocodice.
3. Valutare la complessità dell'algoritmo spiegando il risultato indicato.

Soluzione: usare la visita BFS(u,NOME(x)) modificata, con nodo iniziale x; quando si visita un nuovo nodo, se la distanza assegnata dalla BFS è almeno k, si verifica che NOME del nodo visitato sia uguale a quello di x.

ALGORITMICA Verifica del 16 Maggio 2002

COGNOME **NOME** **CORSO**

Esercizio 3. *(10 punti)* È dato un albero binario T in cui ciascun nodo è colorato di rosso oppure di nero. Ogni nodo è composto di tre attributi: oltre a LEFT e RIGHT che puntano al figlio sinistro e al figlio destro, rispettivamente, l'attributo COLOR memorizza il colore del nodo. Per un arbitrario intero k, si vuole verificare se esiste un sottoalbero in T di k nodi tutti colorati di rosso.

1. Descrivere a parole la struttura dell'algoritmo di risoluzione.
2. Dare una realizzazione dell'algoritmo in pseudocodice.
3. Valutare la complessità dell'algoritmo spiegando il risultato indicato.

Soluzione

```
verificarosso( T, k )
  trovato <- false  // variabile visibile e aggiornabile in rosso()
  if (k <= 0) then return false
  rosso( radice[T], k )
  return trovato

rosso( u, k )
  if u = nil then return 0
  if color[u] = nero then return -1
  l <- rosso( left[u], k )
  r <- rosso( right[u], k )
  if (l < 0) or (r < 0) then return -1  // c'e' almeno un nodo nero
  if (l + r + 1 = k) then trovato <- true
  return (l + r + 1)
```