Algoritmi e Strutture Dati

a.a. 2014/15

Compito del 27/05/2015

Cognome: _____ Nome: _____

Matricola: _____ E-mail: _____

Parte I

(30 minuti; ogni esercizio vale 2 punti)

1. Completare la seguente tabella indicando la complessità delle operazioni che si riferiscono a un dizionario di n elementi. Per l'operazione **Successore** si assuma di essere sull'elemento x a cui si applica l'operazione.

	Ricerca	**Massimo**	**Successore**	**Minimo**	**Costruzione**
Albero binario di ricerca					
Min-Heap					

2. Si confrontino le seguenti funzioni utilizzando le relazioni O, Ω e Θ (giustificando le risposte):

$f(n)$	$g(n)$
$10n^3 + \log n + 2n$	$n^2 + \log n^5$
$n!$	2^n
2^n	2^{n+100}

3. Il problema di determinare i cammini minimi tra tutte le coppie di vertici di un grafo pesato $G=(V, E, w)$ con pesi positivi si può risolvere iterando $|V|$ volte l'algoritmo di Dikstra oppure utlizzando l'algoritmo di Floyd-Warshall. Si riempia la tabella sottostante, specificando le complessità degli algoritmi indicati in funzione della tipologia di grafo utilizzato:

	Grafo sparso	Grafo denso
Iterated_Dijkstra (array)		
Iterated_Dijkstra (heap)		
Floyd-Warshall		

Supponendo che il grafo sia aciclico, quale algoritmo conviene usare? Perché?

Algoritmi e Strutture Dati

a.a. 2014/15

Compito del 27/05/2015

Cognome: _____ Nome: _____

Matricola: _____ E-mail: _____

Parte II

(2.5 ore; ogni esercizio vale 6 punti)

1. Dato un albero binario, i cui nodi sono colorati di *bianco*, *rosso* o *verde*:
   a. scrivere una funzione **efficiente** che stabilisca se esiste un cammino di tre nodi nell'albero i cui colori formano la bandiera italiana. (Il cammino può partire da un nodo qualsiasi, non necessariamente dalla radice.)
   b. Fornire la chiamata della funzione dal programma principale.
   c. Analizzare la complessità di tale funzione.

   Il tipo **Node** utilizzato per rappresentare l'albero binario è il seguente:

```
typedef struct node{
        char * colore;
        struct node * left;
        struct node * right;
} * Node;
```

   Per l'esame da **12 CFU**, deve essere fornita **una funzione C**.
   Per l'esame da **9 CFU**, è sufficiente specificare lo pseudocodice.

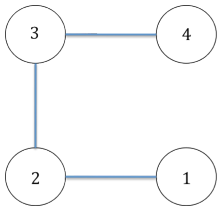
2. Nell'ipotesi di indirizzamento aperto, scrivere uno pseudocodice per HASH-DELETE e modificare HASH-INSERT per gestire il valore DELETED. Analizzare la complessità delle due procedure nel caso pessimo.

3. Il seguente algoritmo accetta in ingresso la matrice di adiacenza di un grafo non orientato e restituisce un valore Booleano (TRUE / FALSE)

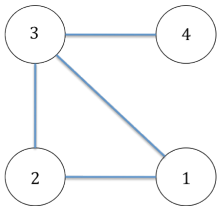
```
MyAlgorithm( A )
 1.   n = rows(A)            /* determina il numero di vertici del grafo */
 2.   let B an n x n matrix   /* alloca memoria per una matrice B n x n */
 3.   for i = 1 to n
 4.      for j = 1 to n
 5.         b[i,j] = 0
 6.         for k = 1 to n
 7.            b[i,j] = b[i,j] + a[i,k]*a[k,j]
 8.   for i = 1 to n
10.      for j = i + 1 to n
11.          if a[i,j]*b[i,j] <> 0 then
12.             return FALSE
13.   return TRUE
```

   Si calcoli la complessità computazionale di `MyAlgorithm` e si determini la sua funzione (in quali casi restituisce TRUE? in quali casi FALSE?). (Nota: Nel determinare la complessità si ignori per comodità la complessità delle istruzioni 1 e 2).

   Si simuli inoltre il suo comportamento sui seguenti due grafi, verificando che restituisca il risultato atteso:



G_1



G_2

4. Sia $G = (V, E)$ un grafo non orientato, connesso e pesato e sia $(S, V \setminus S)$ un taglio di G. Si supponga che l'arco leggero che attraversa il taglio sia unico e lo si denoti con (u,v). In altri termini, per tutti gli altri archi (x,y) che attraversano il taglio, si avrà $w(u,v) < w(x,y)$. Si dimostri che *tutti* gli alberi di copertura minima di G dovranno necessariamente contenere l'arco (u,v).