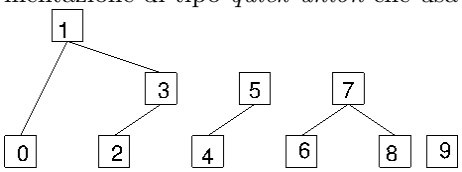
Complementi di Algoritmi e Strutture Dati

(III anno Laurea Triennale - a.a. 2016/17)

Prova scritta 7 giugno 2017

NB: I punteggi sono indicativi.

Esercizio 1 - Union find (punti 6) Considerare la foresta union-find sottostante e un'implementazione di tipo *quick-union* che usa *union-by-size*.



1. Indicare, per ogni radice, il "size" del corrispondente albero.
2. Eseguire nell'ordine le seguenti operazioni (ogni operazione va eseguita sul risultato della precedente):
   union(7,5)
   union(3,6)
   union(2,9)
   Per ogni operazione, spiegare che cosa viene fatto e perché, disegnare il nuovo albero e indicare il "size" della sua radice.
3. Considerare l'esecuzione delle operazioni di cui al punto precedente se l'implementazione adotta la *compressione dei cammini*. Se cambia qualcosa, dire che cosa e perché; se non cambia niente, dire perché.

Esercizio 2 - Sorting (punti 5) Si consideri il seguente array:

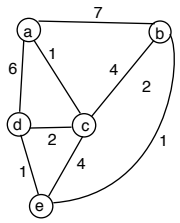
30	10	5	15	20	60	24	48	2

Eseguire la prima fase dell'algoritmo heapsort, cioè quella che trasforma l'array in uno heap a massimo. Si chiede di eseguirla

- PREFERIBILMENTE con la procedura *heapify*.
  Ad ogni passo disegnare tutto l'array come albero ed indicare quali sotto-parti sono già heap.
- IN ALTERNATIVA (ma allora l'esercizio vale un punto in meno) con una serie di chiamate a *insert*.
  Per ogni chiamata indicare l'elemento che viene inserito e disegnare lo heap in cui viene inserito prima e dopo l'operazione (ovviamente il "dopo" di un inserimento è il "prima" dell'inserimento seguente e non occorre ripetere il disegno).

Ricordare che deve essere uno heap *a massimo*.

Esercizio 3 - Grafi (Punti 8) Si esegua, sul seguente grafo:



l'algoritmo di Prim a partire dal nodo a. Inizialmente quindi si avrà `dist`(a)=0, `dist`(b)=∞, `dist`(c)=∞, `dist`(d)=∞, `dist`(e)=∞. Per ogni iterazione del ciclo while si dia:

- il nodo che viene estratto con la `getMin`
- i nodi per i quali viene modificata `dist` e come
- il minimo albero ricoprente alla fine dell'iterazione, evidenziando chiaramente la parte di albero definitiva.

Non dovete disegnare lo heap.

Esercizio 4 - Design e analisi di algoritmi (Punti 7) Si consideri un array $A[1..n]$ in cui ogni elemento può essere esclusivamente 0 oppure 1.

1. Scrivere un algoritmo ottimo che ordini l'array A spostando tutti i valori 0 prima di tutti i valori 1. L'algoritmo deve essere sul posto, basato su scambi (`swap(i,j)` per $1 \leq \mathtt{i}, \mathtt{j} \leq n$) e deve esaminare ogni elemento una volta sola.
2. Giustificare la correttezza dell'algoritmo proposto utilizzando un'opportuna invariante.
3. Analizzare il costo computazionale dell'algoritmo proposto.

Esercizio 5 - NP-completezza (Punti 7) Assumendo $\mathsf{P} \neq \mathsf{NP}$, si dica se le seguenti affermazioni sono vere o false, giustificando la risposta.

1. $\{0,1\}^\star \in \mathsf{P}$
2. esistono linguaggi NP-completi che sono regolari
3. se $\mathcal{P}$ contiene un sottoinsieme NP-completo, allora $\mathcal{P}$ è NP-completo
4. ogni problema NP-completo può essere risolto in tempo $O(2^{p(n)})$ per un qualche polinomio p
5. l'halting problem è NP-completo
6. l'halting problem è NP-hard.