Prof. L. Margara **Scritto del 15/07/2019**

Algoritmi e Strutture Dati

Esercizio 1.[12 punti]
Fornire un algoritmo greedy polinomiale (descrivendo prima l'idea a parole e poi fornendo lo pseudocodice) per risolvere "Max-Clique" (dato un grafo non pesato e non orientato calcolare la cardinalità della clique più grande contenuta nel grafo. Una clique è un sottoinsieme di nodi del grafo a due a due adiacenti). Dimostrare che l'algoritmo fornito non è ottimo. Determinare il suo costo computazionale.

Esercizio 2.[9 punti]
Sia V un min-heap di n elementi. (V è un vettore nel quale è memorizzato il min-heap). Fornire l'algoritmo ottimo per ordinare V. Dimostrare che l'algoritmo proposto è ottimo. Giustificare tutte le risposte.

Esercizio 3.[12 punti]

Sia V un vettore disordinato di n interi positivi. Sia R un vettore disordinato di m interi positivi con $m \leq n$. Fornire il miglior algoritmo possibile (e calcolare il suo costo computazionale) che, presi in input V e R, restituisca un vettore W tale che $W(i) = 1$ se V contiene $R(i)$ e $W(i) = 0$ altrimenti, nei sequenti 3 casi:
1. m costante (non dipende da n);
2. $m = n$;
3. $m = \log(n)$.

Traccia della soluzione dell'esercizio 1.
L'esercizio in questione può essere risolto in molti modi diversi. Qui di seguito riportiamo una possibile soluzione.

Clique(G)
S={un nodo qualsiasi di G}
Q={tutti gli altri nodi di G}
while $Q \neq \emptyset$
    ricercare in Q un nodo collegato
    a tutti i nodi in S.
    Se esiste estrarlo da Q e inserirlo in S,
    Se non esiste ritornare S.

Il costo computazionele di *Clique* è evidentemente polinomiale...
Controesempio: sia $G = (\{1, 2, 3, 4\}, \{(1, 2), (2, 3), (3, 4), (4, 2)\})$

La clique prodotta dall'algoritmo ({1,2}) partendo dal nodo 1 ha cardinalità 2
La clique ottima ({2,3,4}) ha cardinalità 3

Traccia della soluzione dell'esercizio 2.
L'algoritmo ottimo è un qualunque algoritmo di ordinamento con costo $\Theta(n \log(n))$ come ad esempio il Merge-Sort. Si dimostra la sua ottimalità, ad esempio, come abbiamo fatto in classe ovvero facendo vedere che l'ultimo livello dell'Heap può essere un vettore qualunque di lunghezza $n/2$ ...
Altro modo di dimostrare l'ottimalità è notando che è possibile costruire un heap in tempo lineare. Ciò implica che per ordinarlo abbiamo bisogno di tempo almeno $\Theta(n \log(n))$.

Traccia della soluzione dell'esercizio 3.
Caso 1. m costante
Si fanno m ricerche all'interno di V ognuna delle quali costa $\Theta(n)$. Costo totale $\Theta(m\,n) = \Theta(n)$.
Caso 2. $m = n$
Applicando la soluzione del caso 1 avremmo un costo totale $\Theta(m\,n) = \Theta(n\,n) = \Theta(n^2)$. Conviene Ordinare il vettore V (costo $\Theta(n\ \log(n))$) e poi ricercare (ricerca binaria) ogni elemento di R in V. Ogni ricerca costa $\Theta(\log(n))$. Costo totale $\Theta(n\ \log(n) + m\ \log(n)) = \Theta(n\ \log(n))$.
Caso 3. $m = \log(n)$
Applicando la soluzione del caso 1 avremmo un costo totale $\Theta(m\,n) = \Theta(n\ \log(n))$. Inutile ordinare V perché già questa operazione costerebbe $\Theta(n\ \log(n))$. Se invece ordiniamo R spendiamo $\Theta(\log(n)\ \log(\log(n)))$ e poi spendiamo $\Theta(\log(\log(n)))$ per ogni ricerca (ne facciamo n). Costo totale $\Theta(\log(n)\ \log(\log(n)) + n\ \log(\log(n))) = \Theta(n\ \log(\log(n)))$. Questa è la soluzione più conveniente