Esercizi svolti a lezione

Problema 1

In un corso di laurea sono previsti un certo numero di esami obbligatori. Esistono inoltre dei vincoli di propedeuticità: se un esame A è propedeutico ad un esame B allora B deve essere sostenuto in una sessione d'esami successiva a quella in cui è stato sostenuto A. Se due o più esami non violano alcun vincolo di propedeuticità allora possono essere sostenuti tutti all'interno della stessa sessione. Supponendo di superare ogni esame al primo tentativo fornire un algoritmo lineare che calcoli il minimo numero di sessioni necessarie per sostenere tutti gli esami. Per ogni sessione d'esami, fornire inoltre l'insieme di esami da sostenere.
Suggerimento: Considerare il grafo delle propedeuticità e le sue proprietà strutturali.

Problema 2

Ci sono tre contenitori d'acqua di capacità intere pari a c_1, c_2 e c_3 litri, rispettivamente, con $c_1 \geq c_2 \geq c_3 > 0$. Inizialmente il primo contenitore (di capacità c_1) è completamente pieno, mentre i due restanti sono vuoti. L'unica operazione ammissibile consiste nel versare l'acqua di un contenitore A in un altro contenitore B a patto che alla fine dell'operazione valga almeno una tra le seguneti condizioni:

- Il contenitore A è completamente vuoto.
- Il contenitore B è completamente pieno.

Non è consentito sprecare acqua. Dati 3 interi $\ell_1, \ell_2, \ell_3 \geq 0$, progettare un algoritmo che stabilisca se esiste una serie di operazioni che porta il primo contenitore a contenere esattamente ℓ_1 litri, il secondo a contenere ℓ_2 litri ed il terzo a contenerne ℓ_3. In caso affermativo restituire la più corta sequenza di operazioni in grado di raggiungere lo scopo. Il tempo richiesto dall'algoritmo deve essere polinomiale nel valore di c_1.
Suggerimento: Modellare la situazione come un problema su grafi.

Problema 3

Dato un grafo diretto aciclico (DAG) $G = (V, E)$ e due nodi $s, t \in V$ progettare un algoritmo lineare (in $|V|$ e $|E|$) in grado di trovare il più lungo cammino tra s e t.

Problema 4

È dato un grafo $G = (V, E)$. Si vogliono posizionare delle monete sui vertici di G. Per posizionare una moneta, questa deve dapprima essere piazzata su di un vertice x di G non ancora occupato e successivamente spostata su un vertice adiacente ad x, anch'esso non occupato. Progettare un algoritmo lineare che trovi il modo di posizionare il maggior numero di monete possibili sui vertici di G.
Suggerimento: Pensare al caso in cui G è un albero ed estendere la soluzione trovata a grafi generici.

Problema 5

Dato un grafo $G = (V, E)$ con pesi positivi sugli archi ed un insieme di k centri $C = \{c_1, c_2, \ldots, c_k\} \subseteq V$, si richiede di partizionare l'insieme V in k insiemi $V_1, V_2, \ldots, V_k$ in modo che tutti i vertici in un insieme V_i siamo più vicini al vertice c_i che ad ognuno degli altri centri in C. Formalmente, se $d(u, v)$ indica la distanza tra i vertici u e v in G, deve valere:

$$d(u, c_i) \leq d(u, c_j)\ \forall i, j = 1, \ldots, k\ \forall u \in V_i$$

Progettare un algoritmo che risolva tale problema in tempo $O(|E|+|V| \log |V|)$.

Problema 6

Dato un albero $T = (V, E)$ radicato in r ed un insieme $S \subseteq V \times V$ di coppie di nodi, progettare un algoritmo che, per ogni coppia $(u, v) \in S$ resituisca il minimo antenato comune tra u e v in V.
Nota: Il minimo antenato comune tra due nodi u e v ($\mathrm{LCA}(u, v)$) è il più profondo nodo che appartiene sia all'unico cammino tra u ed r in T che all'unico cammino tra v ed r in T.
Soluzione:

Lo pseudocodice della soluzione è riportato di seguito.

Algorithm 1: Minimo Antenato Comune

```
procedure LCA(T, r)
    for ogni nodo v di T do
        MakeSet(v)
        Color[v] ← White

    Visita(T, r)

procedure Visita(T, u)
    for ogni figlio v di u in T do
        Visita(T, v)  Union(u, v)

    Color[u] ← Black

    for ogni coppia (u, v) ∈ S do
        if Color[v] = Black then
            Print Il minimo antenato comune tra u e v è Find(v)
```

L'idea consiste nell'effettuare una visita in profondità in postordine dell'albero T memorizzando, per ogni nodo u, il meno profondo antenato di u già visitato. Per fare ciò, l'algoritmo fa uso di una struttura Union-Find in cui, inizialmente, ogni insieme corrisponde ad un singolo nodo di T. Quando viene visitato un nodo, tutti gli insiemi associati ai suoi figli vengono uniti con l'insieme che ha per rappresentante u. I nodi visitati vengono marcati facendo uso del vettore Color.

Prima che la visita di un nodo u abbia termine, si esaminano tutte le coppie (u, v) in cui compare u: se anche il nodo v è già stato visitato ciò significa che la visita ha attraversato gli archi dell'albero tra v e $\text{LCA}(u, v)$, e quindi gli archi tra $\text{LCA}(u, v)$ ed u. Dal momento che $LCA(u, v)$ è ad una profondità inferiore (o uguale) rispetto ad u e v in T, ciò significa che il rappresentante dell'insieme contenente v è proprio $\text{LCA}(u, v)$, come riportato dall'algoritmo.

Per stimare la complessità computazionale, notiamo che l'ultimo ciclo può essere eseguito in un tempo complessivo di $O(|S|)$ semplicemente assegnando ad ognuno dei nodi, una lista delle coppie in cui compare. Le operazioni relative alla visita in profondità richiedono tempo $O(|V|)$.

Resta da calcolare la complessità relativa alle $|V|$ operazioni MakeSet, alle $|V| - 1$ operazioni Union, ed alle $|S|$ operazioni Find.

La migliore implementazioni conosciuta delle struttura Union-Find consente di eseguire n MakeSet in tempo $O(n)$, ed m tra Union e Find in tempo $O(m\alpha(n))$ in cui $\alpha(n)$ indica la funzione inversa della funzione di

Ackermann, la cui crescita è estremamente lenta ($\alpha(n) \leq 4$ per qualsiasi valore pratico di n).

Nel nostro caso il tempo complessivamente richiesto delle operazion di MakeSet, Union e Find è $O((|S|+|V|)\alpha(|V|))$, che coincide (asintoticamente) con il tempo richiesto per l'esecuzione dell'intero algoritmo.

Problema 7

Sia $G = (V, E)$ un grafo con pesi positivi sugli archi tale che $V = \{s, t\} \cup V_1 \cup V_2 \cup \cdots \cup V_k$ in cui tutti i precedenti insiemi sono a due a due disgiunti. Progettare un algoritmo in grado di trovare il più breve cammino π tra s e t passante per almeno un nodo di ogni insieme V_i, in ordine. L'algoritmo deve avere complessità $O(k(|E| + |V| \log |V|))$.
Nota: Non è necessario che i nodi degli insiemi V_i siano consecutivi in π. I sottocammini tra nodi di insiemi consecutivi (siano essi V_i e V_{i+1}) possono attraversare qualsiasi vertice del grafo (anche vertici di insiemi V_j con $j < i$ o $j > i + 1$).

Problema 8

Dato un grafo $G = (V, E)$ connesso e con pesi interi sugli archi nell'intervallo $[1, k]$, progettare un algoritmo in grado di calcolare un albero dei cammini minimi di G in tempo $O(k|E|)$.

Problema 9

Sia $G = (V, E)$ un grafo con pesi positivi distinti sugli archi ed $e \in E$ un arco di G. Progettare un algoritmo lineare in grado di determinare se esiste un MST di G che contiene l'arco e.

Problema 10

Si vuole organizzare una festa con il maggior numero possibile di invitati in modo che ognuno di essi conosca almeno k persone e, al contempo, non ne conosca almeno k. Dato l'intero k, l'insieme di potenziali invitati S ed una lista di tutte le conoscenze tra coppie di persone in S, si richiede di progettare un algoritmo polinomiale in grado di determinare l'insieme $I \subseteq S$ di persone da invitare. L'insieme I deve soddisfare i precedenti vincoli e deve contenere il maggior numero possibile di persone.

Problema 11

Dato un grafo $G = (V, E)$ connesso e pesato sugli archi ed un insieme $U \subseteq V$ di nodi, progettare un algoritmo che calcoli uno spanning tree T di G di costo

minimo con il vincolo che tutti i nodi in U siano foglie di T. L'algoritmo deve avere complessità $O(|E| \log |V|)$.