Computabilità ed Algoritmi
Esempio di Esame di Algoritmi
25 Maggio 2018

1. Sia G un grafo non orientato. Supponiamo di iniziare con due monete disposte su due vertici arbitrariamente scelti di G. Ad ogni passo, ogni moneta *deve* spostarsi su un vertice adiacente.

Descrivere e analizzare un algoritmo efficiente per calcolare il numero minimo di passaggi per raggiungere una configurazione in cui entrambe le monete si trovano sullo stesso vertice o per segnalare che tale configurazione non è raggiungibile.

L'input dell'algoritmo consiste in un grafo $G = (V, E)$ e due vertici $u, v \in V$ (che possono o non possono essere distinti).

Soluzione: Sia $G = (V, E)$ il grafo di input, e siano s e t le posizioni iniziali delle monete. Modelliamo il problema con un grafo non orientato e non pesato $G' = (V', E')$:

- $V' = V \times V = \{(u, v) \mid u, v \in V\}$: i vertici di G' corrispondono alle posizioni possibili delle due monete;
- Gli archi corrispondono alle mosse valide: $E' = \{\{(u, v), (u', v')\} \mid \{u, u'\} \in E \text{ e } \{v, v'\} \in E\}$. Gli archi non sono orientati perché ogni mossa può essere eseguita in entrambe le direzioni.
- I vertici sono associati alle posizioni delle monete come specificato. Non serve aggiungere informazione agli archi.
- Il problema da risolvere è trovare il cammino minimo da (s, t) ad un qualsiasi vertice del tipo (v, v) dove le due monete sono nella stessa posizione.
- L'algoritmo da usare per risolvere il problema è BFS. La visita in ampiezza inizia dal vertice (s, t). Al termine della vista una semplice iterazione su tutti i vertici (v, v) ci permette di trovare la lunghezza del cammino minimo. Se nessun vertice (v, v) è stato visitato, allora il cammino non esiste.
- Il tempo impiegato per costruire il grafo G' è proporzionale alla sua dimensione. Se il grafo G ha n vertici e m archi, allora G' ha n^2 vertici e al più m^2 archi. Quindi la sua costruzione ha costo $O(n^2 + m^2)$. La complessità di BFS è $O(|V'| + |E'|) = O(n^2 + m^2)$, mentre l'iterazione finale per trovare il vertice a distanza minima costa $O(n)$. Quindi l'algoritmo complessivamente impiega tempo $O(n^2 + m^2)$.

2. Hai appena scoperto il tuo migliore amico della scuola elementare su Twitbook. Volete incontrarvi il prima possibile, ma vivete in due diverse città distanti tra di loro. Per ridurre al minimo i tempi di viaggio, stabilite di incontrarvi in una città intermedia, e poi salite contemporaneamente in auto per guidare l'uno verso l'altro. Ma dove dovreste incontrarvi esattamente?

Supponente di avere un grafo pesato $G = (V, E)$, in cui i vertici V rappresentano le città e gli archi E rappresentano le strade che collegano direttamente le città. Ogni arco $\{u, v\}$ ha un peso $w(u, v)$ uguale al tempo richiesto per viaggiare tra le due città. Vi viene fornito anche un vertice p, che rappresenta la tua posizione di partenza, e un vertice q, che rappresenta la posizione di partenza del tuo amico.

Descrivi e analizza un algoritmo per trovare il vertice intermedio che consente a te e al tuo amico di incontrarvi il prima possibile, assumendo che partiate entrambi da casa *nello stesso momento*.

Sia $dist(x, y)$ è il tempo di percorrenza del percorso più breve dal vertice x al vertice y. Se io e il mio amico decidiamo di incontrarci nel vertice intermedio t, e partiamo dai nostri rispettivi vertici al tempo 0, allora ci incontreremo al tempo $\max\{dist(p, t), dist(q, t)\}$. Possiamo calcolare questo tempo per tutti i vertici t eseguendo l'algoritmo di Dijkstra due volte, una volta da p e una volta da q, e quindi iterando su tutti i vertici. Il tempo di esecuzione complessivo è $O(|E| \log |V|)$.

```
function WhereToMeet(G, p, q)
    distp = Dijkstra(G, p)        ▷ distp: distanza del cammino minimo da p a t
    distq = Dijkstra(G, q)        ▷ distp: distanza del cammino minimo da q a t
    time = ∞; best = NULL
    for t ∈ V do
        if time > max{distp[t], distq[t]} then
            time = max{distp[t], distq[t]}
            best = t
    return best
```

3. Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo arbitrario contiene un circuito Hamiltoniano è un problema NP-hard. Anche stabilire se il grafo contiene un *cammino* Hamiltoniano è NP-hard.

Sia G un grafo non orientato e pesato. Un circuito Hamiltoniano di G è *pesante* se il peso totale degli archi nel ciclo è almeno la metà del peso totale di tutti gli archi di G. Dimostrare che stabilire se un grafo contiene un ciclo Hamiltoniano pesante è NP-hard.

Soluzione: Il problema NP-hard scelto per la riduzione è quello del *cammino* Hamiltoniano. Sia $G = (V, E)$ un grafo non orientato e non pesato e supponiamo che G abbia n vertici. Costruiamo un grafo non orientato e pesato H aggiungendo due nuovi vertici s e t a G. I due nuovi vertici sono collegati con tutti gli altri vertici di H da un arco (compreso l'arco $\{s, t\}$). Tutti gli archi hanno peso 0 tranne l'arco $\{s, t\}$ che ha peso 1. Dimostriamo che G possiede un cammino Hamiltoniano se e solo se H ha un circuito Hamiltoniano pesante.

- $\Rightarrow$ Supponiamo che G possieda un cammino Hamiltoniano P. Quindi P parte da un nodo u e termina in un nodo v visitando tutti i vertici di G. Possiamo costruire un circuito Hamiltoniano pesante per H aggiungendo gli archi $\{v, s\}$, $\{s, t\}$ e $\{t, u\}$ a P: il circuito visita tutti i vertici di H e ha peso 1. Poiché il peso totale degli archi in H è 1, allora il circuito è pesante.
- $\Leftarrow$ Supponiamo che H abbia un circuito Hamiltoniano pesante C. Questo ciclo deve necessariamente contenere l'arco $\{s, t\}$, altrimenti avrebbe peso 0 e non sarebbe pesante. Quindi devono esistere due vertici u, v di G tali che gli archi $\{u, s\}$ e $\{t, v\}$ appartengono a C. Se eliminamo gli archi $\{u, s\}$, $\{t, v\}$ e $\{s, t\}$ da C otteniamo un cammino da u a v che visita tutti i vertici di G, cioè un cammino Hamiltoniano.

Per concludere, dato G per costruire H dobbiamo aggiungere 3 nuovi vertici e $n + 1$ archi, e assegnare i pesi agli archi. Questa operazione si può eseguire in tempo polinomiale.