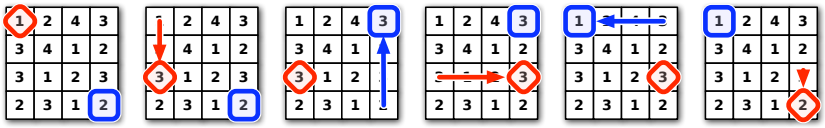
Computabilità ed Algoritmi
Esame di Algoritmi
17 Settembre 2018

Tempo a disposizione: 2 h 00 min

1. Considerate il seguente rompicapo, che si gioca su una griglia quadrata $n \times n$ con due pedine, una rossa e una blu. Ogni cella della griglia è etichettata con un numero intero positivo. Le pedine si trovano sempre su caselle distinte della griglia. All'inizio del gioco le pedine si trovano negli angoli in alto a sinistra e in basso a destra della griglia; l'obiettivo del rompicapo è scambiare di posizione le pedine.

In ogni turno, puoi muovere una pedina in alto, a destra, in basso o a sinistra di una *distanza determinata* ***dall'altra pedina***. Ad esempio, se la pedina rossa si trova su un quadrato etichettato con 3, puoi spostare la pedina blu 3 passi in alto, 3 passi a sinistra, 3 passi a destra o 3 passi in basso. Non puoi mai spostare una pedina fuori dalla griglia o nello stesso quadrato dell'altra pedina.



Una soluzione in 5 mosse di un rompicapo 4×4.

Considerate il problema di trovare il numero minimo di mosse necessarie per risolvere un rompicapo, o di riportare correttamente che il rompicapo non ha soluzione. Rappresentate il problema con un grafo e applicate uno degli algoritmi visti a lezione per risolverlo.

Fornite le seguenti informazioni:

- Quali sono i vertici? Cosa rappresenta ciasc vertice?
- Quali sono gli archi? Sono orientati o non orientati?
- Se i vertici e/o gli archi hanno dei valori associati, quali sono questi valori?
- Quale problema si deve risolvere sul grafo?
- Quale algoritmo si può usare per risolvere il problema?
- Qual è il tempo di esecuzione dell'intero algoritmo, *incluso* il tempo di creazione del grafo, *in funzione dei parametri di input originali*?

Chiamiamo M la matrice $n \times n$ che rappresenta le celle della griglia, e modelliamo il problema con un grafo non orientato e non pesato $G = (V, E)$:

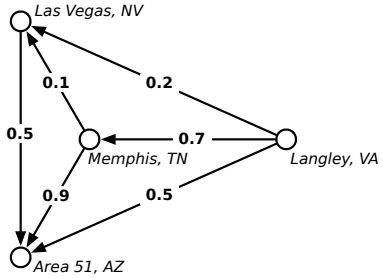
- I vertici di G sono quadruple (x_1, y_1, x_2, y_2) che rappresentano le posizioni delle due pedine. Il grafo comprende solo i vertici con posizioni valide: $(x_1, y_1) \neq (x_2, y_2)$ (le due pedine sono su posizioni diverse). I valori di x_1, y_1, x_2, y_2 variano tra 0 e $n - 1$.
- Gli archi corrispondono alle mosse valide. Nel caso in cui sia la pedina in (x_1, y_1) a muovere mettiamo un arco da (x_1, y_1, x_2, y_2) a (x_1', y_1', x_2', y_2') se $(x_2, y_2) = (x_2', y_2')$ (la seconda pedina rimane ferma) e si verifica una delle seguenti condizioni:
  - $x_1' = x_1 + M[x_2, y_2]$ e $y_1' = y_1$ (mossa verso destra);
  - $x_1' = x_1 - M[x_2, y_2]$ e $y_1' = y_1$ (mossa verso sinistra);
  - $x_1' = x_1$ e $y_1' = y_1 + M[x_2, y_2]$ (mossa verso l'alto);
  - $x_1' = x_1$ e $y_1' = y_1 - M[x_2, y_2]$ (mossa verso il basso).

  Nel caso in cui sia la pedina in (x_2, y_2) a muovere le condizioni sono analoghe: basta scambiare di ruolo le posizioni delle pedine nei casi precedenti. Gli archi sono non orientati (le mosse sono sempre reversibili) e non pesati.
- I vertici sono associati alle posizioni delle pedine come specificato. Gli archi non hanno informazione aggiuntiva.
- Il problema da risolvere è trovare il cammino minimo dal vertice che corrisponde alla posizione iniziale $(0, 0, n - 1, n - 1)$ verso il vertice che corrisponde alla posizione finale $(n - 1, n - 1, 0, 0)$.
- Poiché gli archi sono privi di peso, il problema si può risolvere con una visita in ampiezza del grafo (BFS) partendo dalla posizione iniziale.
- Il tempo impiegato per costruire il grafo è proporzionale alla sua dimensione. Il numero dei vertici dipende dai valori che le variabili x_1, y_1, x_2, y_2 possono assumere. Da ogni vertice partono al più otto archi, quindi il grafo possiede al più n^4 vertici e al più $8n^4$ archi. Durante

la costruzione, ogni volta che si aggiunge un arco al grafo si deve accedere alla matrice M per stabilire se il nuovo arco è valido. L'accesso alla matrice ha costo $O(1)$, quindi la costruzione del grafo ha un costo asintotico di $O(n^4)$. La complessità di BFS è $O(|V| + |E|) = O(n^4)$. Quindi l'intero algoritmo impiega tempo $O(n^4)$.

2. Mulder e Scully hanno calcolato, per ogni strada negli Stati Uniti, l'esatta probabilità che qualcuno che guidi su quella strada *non venga* rapito dagli alieni. L'agente Mulder ha bisogno di andare da Langley, Virginia, all'Area 51, Nevada. Che percorso dovrebbe prendere in modo che abbia la minima probabilità di essere rapito?

Più formalmente, vi viene dato un grafo orientato $G = (V, E)$, dove ogni arco (u, v) ha una probabilità di sicurezza $p(u, v)$. La *sicurezza* di un cammino è il prodotto delle probabilità di sicurezza dei suoi archi. Assumendo che le probabilità siano indipendenti, progettare e analizzare la complessità di un algoritmo per determinare il percorso più sicuro da un dato vertice di partenza s a un dato vertice di destinazione t.



Per esempio, con le probabilità mostrate sopra, se Mulder va direttamente da Langley all'Area 51, ha una probabilità del 50% di arrivare senza essere rapito. Se si ferma a Memphis, ha una probabilità pari a $0.7 \cdot 0.9 = 63\%$ di arrivare in sicurezza. Se si ferma prima a Memphis e poi a Las Vegas, ha una probabilità di rapimento pari a $1 - 0.7 \cdot 0.1 \cdot 0.5 = 96.5\%$! Sebbene questo esempio sia un grafo diretto aciclico, il tuo algoritmo deve gestire grafici diretti arbitrari.

Soluzione 1. Per trovare il percorso più sicuro da s a t, si può eseguire l'algoritmo di Dijkstra assegnando agli archi peso $w(u, v) = -\log p(u, v)$ per trovare i cammini minimi a partire da s in tempo $O(E + V \log V)$. Il percorso più sicuro è il percorso di peso minimo da s a t, e la sicurezza di questo percorso è il prodotto delle probabilità di sicurezza degli archi.

Ecco perché questo metodo funziona. La sicurezza di un percorso è il prodotto delle probabilità di sicurezza degli archi che lo compongono. Vogliamo trovare un cammino π da s a t che massimizzi il prodotto $\prod_{(u,v)\in\pi} p(u, v)$. Per le proprietà dei logaritmi, questo è equivalente a massimizzare la sommatoria $\sum_{(u,v)\in\pi} \log p(u, v)$, ossia a minimizzare la sommatoria $\sum_{(u,v)\in\pi} -\log p(u, v)$. Quindi se assegnamo peso $w(u, v) = -\log p(u, v)$ agli archi otteniamo un problema di cammino minimo.

Poiché le probabilità $p(u, v)$ variano tra 0 e 1, allora il loro logaritmo $\log p(u, v)$ è negativo, e quindi non ci sono archi di peso negativo nel grafo. Di conseguenza, possiamo utilizzare l'algoritmo di Dijkstra per risolvere il problema in tempo $O(E + V \log V)$.

Soluzione 2. In alternativa, si può modificare l'algoritmo di Dijkstra in modo che lavori con le probabilità originali, senza dover modificare il grafo originale. La versione modificata dell'algoritmo massimizza il prodotto delle probabilità lungo un percorso invece di ridurre al minimo la somma dei pesi lungo un percorso, e procede come segue:

- le probabilità di sicurezza $p(u, v)$ vengono usate al posto dei pesi degli archi;
- i valori di $d[v]$ vengono inizializzati a $-\infty$ per i vertici $v \neq s$, e a 1 per $d[s]$;
- si usa una Max-Heap al posto di una Min-Heap, ExtractMax) al posto di ExtractMin, IncreaseKey) al posto di DecreaseKey;
- il rilassamento degli archi si effettua come segue:

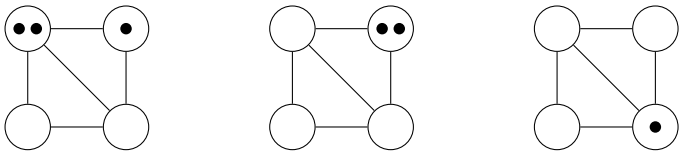
  function Relax(u, v)
  if $d[v] < d[u] \cdot p(u, v)$ **then**
  $d[v] = d[u] \cdot p(u, v)$
  $\pi[v] = u$

L'algoritmo di Dijkstra modificato è isomorfo alla soluzione precedente: esegue le stesse operazioni sul grafo, lavorando con le probabilità originali invece del loro logaritmo.

3. Dimostrare che un problema X è NP-hard richiede diversi passaggi:

- Scegli un problema Y che sai essere NP-hard.
- Descrivi un algoritmo per risolvere Y usando un algoritmo per X come subroutine. Tipicamente questo algoritmo ha la seguente forma: data un'istanza di Y, trasformala in un'istanza di X, quindi chiama l'algoritmo magico black-box per X.
- Dimostra che l'algoritmo è corretto. Ciò richiede sempre due passaggi separati, che di solito hanno la seguente forma:
  - Dimostra che il tuo algoritmo trasforma istanze "buone" di Y in istanze "buone" di X.
  - Dimostra che il tuo algoritmo trasforma istanze "cattive" di Y in istanze "cattive" di X. Equivalentemente: Dimostra che se la tua trasformazione produce un'istanza "buona" di X, allora era partita da un'istanza "buona" di Y.
- Mostra che il tuo algoritmo per Y funziona in tempo polinomiale, a meno della chiamata (o delle chiamate) all'algoritmo magico black-box per X. (Questo di solito è banale.)

Pebbling è un solitario giocato su un grafo non orientato G, in cui ogni vertice ha zero o più ciottoli. Una mossa del gioco consiste nel rimuovere due ciottoli da un vertice v e aggiungere un ciottolo ad un vertice u adiacente a v (il vertice v deve avere almeno due ciottoli all'inizio della mossa). Il problema PebbleDestruction chiede, dato un grafo $G = (V, E)$ ed un numero di ciottoli $p(v)$ per ogni vertice v, di determinare se esiste una sequenza di mosse che rimuove tutti i sassolini tranne uno.



Una soluzione in 3 mosse di PebbleDestruction.

Dimostra che PebbleDestruction è NP-hard usando il problema del Circuito Hamiltoniano come problema NP-hard noto (un circuito Hamiltoniano è un ciclo che attraversa ogni vertice di G esattamente una volta).

Mostriamo come trasformare un'istanza del problema del circuito Hamiltoniano in un'istanza di PebbleDestruction. Sia $G = (V, E)$ un grafo non orientato e non pesato e supponiamo che G abbia n vertici. Assegnamo il numero di ciottoli $p(v)$ degli n vertici come segue: due ciottoli ad vertice arbitrario $s \in V$ ($p(s) = 2$), un ciottolo per ogni altro vertice del grafo ($p(v) = 1$ per ogni $v \neq s$). Dimostriamo che G possiede un cammino Hamiltoniano se e solo se esiste una soluzione di PebbleDestruction per l'assegnamento di ciottoli $p(v)$ dato in precedenza.

- $\Rightarrow$ Supponiamo che G possieda un circuito Hamiltoniano P. Quindi P visita tutti i vertici del grafo, compreso il vertice s con due ciottoli. La sequenza di mosse parte da s e segue gli archi del circuito P rimuove tutti i sassolini tranne uno.
- $\Leftarrow$ Supponiamo che esista una sequenza di mosse che rimuove tutti i ciottoli tranne uno. Questa sequenza deve ciclo deve necessariamente partire dal vertice s, che è l'unico che si può svuotare all'inizio, e deve visitare tutti gli altri vertici del grafo. Quindi forma un circuito Hamiltoniano in G.

Per concludere, dato G per costruire l'istanza di PebbleDestruction dobbiamo assegnare i ciottoli agli n vertici. Queast operazione che si può eseguire in tempo $O(n)$.