TECNICHE DI PROGRAMMAZIONE

21/05/19

Cicli Euleriani e cicli Hamiltoniani.

Ultimo argomento rimasto in sospeso sui grafi che è quello sui cicli.

Un ciclo è un cammino chiuso, un cammino in cui il primo vertice e l’ultimo vertice sono coincidenti. Abbiamo 2 tipi di cicli:

1. Ciclo Hamiltoniano: cicli che toccano tutti i vertici di un grafo (cicli semplici). A volte si parla di “cammino hamiltoniano” quando abbiamo un ciclo aperto in cui non torno al punto di partenza.

Immagine che contiene oggetto

Descrizione generata automaticamente

1. Ciclo Euleriano: cicli in cui bisogna attraversare tutti gli archi di un grafo. Affinché io possa attraversare tutti gli archi, devo passare più volte da uno stesso vertice.

La situazione cambia se ho un grafo pesato. Se ho un grafo non pesato, il problema si risolve trovando un ciclo hamiltoniano o euleriano. I cicli hamiltoniani, se esistono, sono tutti equivalenti tra di loro in quanto non ho dei pesi. Se invece ho un grafo pesato, oltre a cercare un cammino hamiltoniano, devo anche verificare che esso sia quello di peso minimo.

I 2 problemi che sembrano molto simili (toccare tutti gli archi e toccare tutti i vertici) dal punto di vista algoritmico sono completamente diversi.

Partiamo dai cicli Euleriani (toccare tutti gli archi). È presente un algoritmo semplicissimo che ci permette di trovare un grafico un grafo euleriano (se esiste). L’obiettivo dell’algoritmo è quello di partire da un vertice, toccare tutti gli archi e cercare di tornare al punto di partenza. Questo algoritmo funziona se il grado di tutti vertici è pari. Esso consiste di partire da un vertice scelto e muoversi a caso. Il nostro grafo ha un numero pari di vertici e quindi non rischio di rimanere bloccato in un vertice.

Se per esempio prendo un grafo di grado 2 (con 2 vertici e 2 archi), partendo da uno dei due vertici, posso arrivare all’altro attraverso un arco (scelto a caso) e posso tornare al vertice di partenza mediante l’altro arco.

Usando questo algoritmo, posso trovare un ciclo chiuso, ma non è detto che io abbia usato tutti gli archi. Per risolvere questo inconveniente, vado a cercare gli archi che non ho ancora toccato, mi riposiziono sul vertice che ha questo arco libero e provo a compiere un altro tragitto. Esempio:

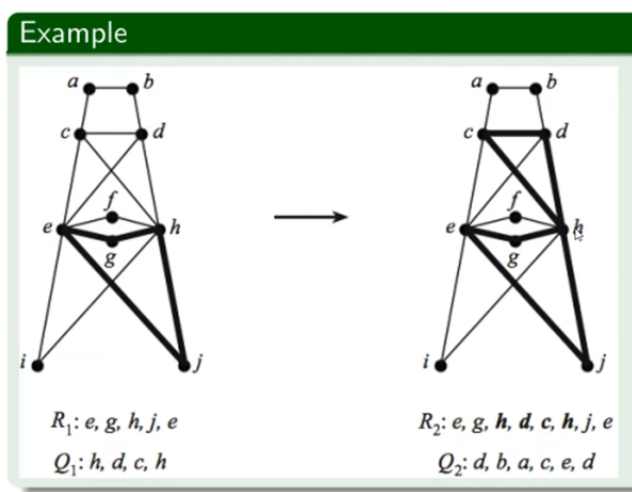


Immagine che contiene oggetto

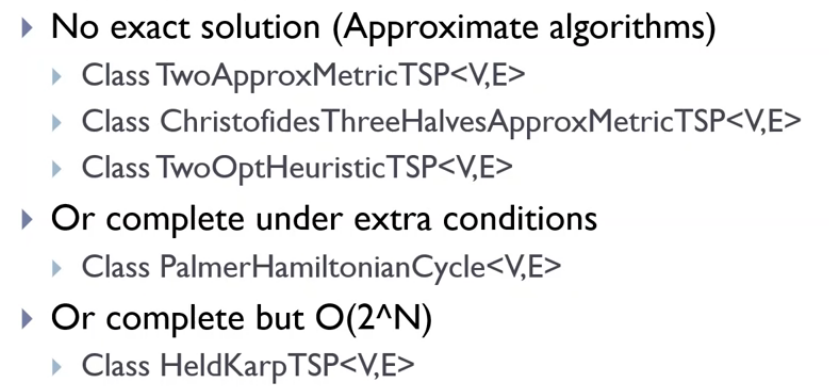
Descrizione generata automaticamente

Condizione fondamentale affinché io possa usare questo teorema è che tutti i vertici devono avere grado pari! Ciò vuol dire che da ogni vertice devono vertice un numero pari di archi.

Questo appena analizzato era il teorema di Eulero. Per quanto riguarda i cicli Hamiltoniani invece, il nostro obiettivo è quello di trovare almeno un ciclo che tocca tutti vertici. Useremo degli algoritmi semplificati che sfruttano la disuguaglianza triangolare.

Quest’ultima ci dice che se io ho 3 punti (A,B,C), il percorso diretto A-C è sempre più veloce rispetto a fare A-B e B-C. Quando affrontiamo i problemi sui cicli Hamiltoniani, ci troveremo di fronte a complessità algoritmiche molto forti (esponenziale, fattoriale).

Sono nati dunque degli “algoritmi approssimati” che ci permettono di approssimare la complessità di un problema. Se andiamo nella documentazione di “jgrapht.org” troviamo un’interfaccia che si chiama “HamiltonianCycleAlgorithm”, che ha una serie di implementazioni possibili. CI sono 3 classi che ci danno una soluzione approssimata, 1 classe che ci da la soluzione completa in caso di condizione extra e un ultima che ci dà la soluzione completa solo se la complessità è O(2^N).



La classe “TwoApproxMetric” ci permette di trovare un cammino che sicuramente è minore del doppio del cammino minimo. Essa funziona solamente quando viene rispettata la disuguaglianza triangolare.

La classe “ChristofidesThreeHalvesApproxMetric” è migliore rispetto alla precedente in quanto il cammino trovato sarà sicuramente minore dei 3/2 del cammino minimo. Questa classe ha come ipotesi che la disuguaglianza triangolare sia soddisfatta e che i pesi degli archi siano tutti non-negativi. Inoltre, il grafo deve essere pesato, non orientato e completo.

Il professore ha poi continuato a leggere sulla documentazione del sito cosa facevano le altre classi (immagine sopra). Consiglio di leggere almeno una volta cosa fanno tutte le classi (non importante).

È stata approfondita ulteriormente la classe “TwoApproxMetric”. Viene consigliata come classe da cui partire in quanto non mi richiede nulla sul grafo (se è completo, pesato, orientato ecc…). L’unica condizione da rispettare è quella della disuguaglianza triangolare. Essa ci ritorna un ordinamento in profondità del “minimum spanning tree”, cioè dell’albero di visita di peso minimo. Il problema è che all’inizio noi abbiamo un albero e non un ciclo (sono due cose completamente differenti). Quindi, l’algoritmo trova l’albero di cammino minimo e poi lo connette.

Quindi l’algoritmo usa questo spanning tree che ha una proprietà minimale per trovare dei segmenti di percorsi che sono minimizzati. Successivamente li collego insieme per trovare un percorso minimo.

FINE