Laboratorio di Algoritmi e Strutture Dati

Prof. Aniello Murano

Componenti fortemente connesse e Alberi minimi di copertura

Corso di Laurea: Informatica
Codice insegnamento: 13917
Email docente: murano@na.infn.it
Anno accademico: 2007/2008

Lezione numero: 21

Parole chiave: Bellman-Ford, Dijkstra, Floyd-Warshall

Riepilogo delle lezioni precedenti

Definizione di Grafo orientato e non orientato

Rappresentazione di Grafi tramite l'suo di liste di adiacenza e di matrici di adiacenza

Implementazione di algoritmi di base per la gestione di grafi (orientati e non orientati):

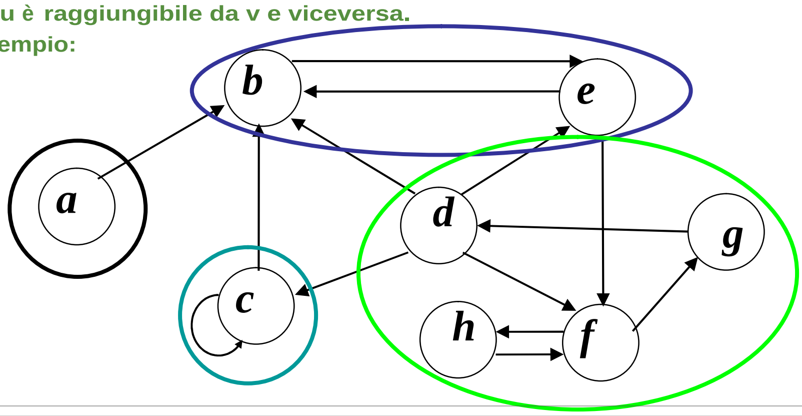
- Creazione di un grafo (semplice o pesato)
- Interrogazione di un grafo: calcola grado, stampa grafo, ecc.)
- Modifica di un grafo: Aggiungi/rimuovi vertice, aggiungi/rimuovi arco, grafo trasposto, ecc.
- Visita di tutti i nodi di un grafo (BFS e DFS) e raggiungibilità di un nodo.
- Calcolo del percorso minimo in un grafo

Oggi vedremo un algoritmo per il calcolo di componenti fortemente connesse, e due algoritmi per il calcolo di un albero minimo di copertura.

Componente fortemente connessa

Dato un grafo diretto G=(V,E), una componente fortemente connessa (SCC, Strongly Connected Component) è un insieme massimale di vertici U sottoinsieme di V tale che per ogni coppia di vertici u e v in U, u è raggiungibile da v e viceversa.

Esempio:



Applicazioni

Dato un grafo, l'operazione di decomposizione nelle sue componenti fortemente connesse ha molte applicazioni pratiche

Per esempio:

- la disposizione di sensi unici in una città,
- I vincoli tra variabili all'interno di un programma, che devono dunque essere presi in considerazione contemporaneamente,
- la correlazione di moduli di un programma e dunque la capacità per un compilatore di raggruppare moduli in modo efficiente

Algoritmo (di Kosaraju) per SCC

L’algoritmo usa il teorema che se si fa una DFS di G e si calcola l’ordine finale di visita dei nodi (f[v]), e poi si rifà una DFS sul grafo trasposto iniziando sempre dai nodi con f massimo, il risultato è una decomposizione di G nelle sue componenti fortemente connesse:

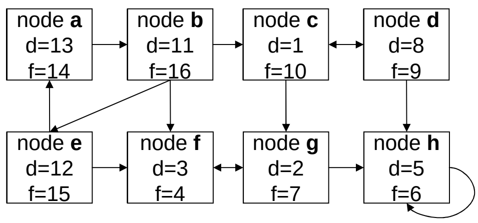
In sintesi, l’algoritmo di Kosaraju per SCC è il seguente:

- Chiamare DFS(G) per computare f[v] (l’ordine finale di visita) per ogni v
- Calcolare il grafo trasposto G’ di G
- Chiamare DFS(G’), ma nel ciclo principale della DFS si considerino i vertici in ordine decrescente di f[v]
- Restituire i vertici di ogni albero calcolato con la DFS(G’) come una SCC

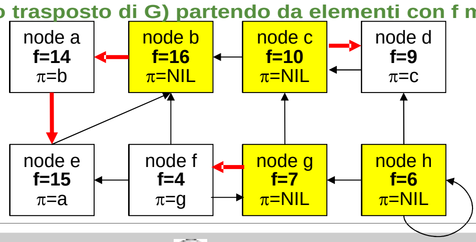
Per calcolare f[v], nella DFS, può essere utile calcolare anche un tempo d relativo a quando v è scoperto (prima volta)

Esempio

DFS su G, partendo da c.

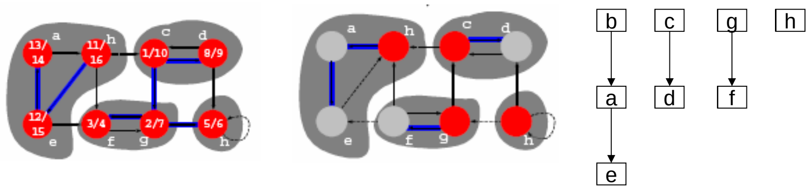


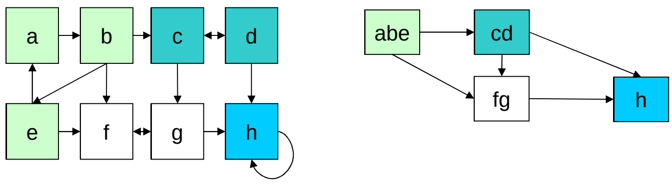
DFS su G’ (il grafo trasposto di G) partendo da elementi con f massima.



Strongly-Connected Components

Di seguito riportiamo i 4 alberi risultanti che formano i quattro SCC del grafo dato.





Esercizio

Implementare in linguaggio C
l'algoritmo di di Kosaraju

Albero ricoprente

Un albero ricoprente di un grafo G=(V,E) è un albero T=(V′,E′) tale che V′ =V e E′ è un sottoinsieme di E.

Si ricorda che un albero è un grafo (non orientato) connesso e senza cicli. Ricordiamo anche che un ciclo in un grafo è un cammino in cui il nodo di partenza e di destinazione coincidono e che non passa due volte per nessun nodo intermedio.

Minimum Spanning Tree

Il problema del *Minimo Albero Ricoprente* **(in breve,** *MST*, **Minimum Spanning Tree) è definito come segue:**

- dato un grafo *G* (non orientato e) pesato, trovare un albero ricoprente di *G* tale che la somma dei pesi dei suoi archi sia minima.

L′importanza di tale problema è enorme. A titolo di esempio, si supponga di dover realizzare l'impianto di distribuzione dell'energia elettrica di una zona, nella quale esistono un certo numero di abitazioni che devono essere servite. Si supponga che di ogni abitazione si conosca la posizione, nonché la distanza da tutte le altre. L'albero di copertura minimo rappresenta la soluzione che consente la minimizzazione della lunghezza dei collegamenti da realizzare.

Nota: L'albero di copertura minimo per un grafo in generale non è unico.

Tecnica greedy per calcolare un MST

Gli algoritmi per il calcolo di un albero di copertura minimo che vedremo, si basano su un approccio chiamato greedy.

Gli algoritmi greedy rappresentano una particolare categoria di algoritmi di ricerca o ottimizzazione.

In molti problemi ad ogni passo l'algoritmo deve fare una scelta: seguendo la strategia greedy (ingordo), l'algoritmo fa la scelta più conveniente in quel momento.

Esempio: Supponiamo di avere un sacchetto di monete di euro e di voler comporre una precisa somma di denaro x, utilizzando il minor numero di monete. La soluzione consiste nel prendere dal sacchetto sempre la moneta più grande tale che sommata alle monete scelte in precedenza il totale non sia superiore a x.

In molti casi (ma non sempre), la tecnica greedy fornisce una soluzione globalmente ottima al problema.

Algoritmo generico per un MST

L'idea generale per calcolare un MST T su un grafo G è la seguente:

```
T=NULL;
while T non forma un albero di copertura;
do trova un arco (u,v) che sia sicuro per T;
T=T unito a {(u,v)};
return T
```

Si definisce sicuro un arco che aggiunto ad un sottoinsieme di un albero di copertura minimo produce ancora un sottoinsieme di un albero di copertura minimo.

In seguito vedremo due tecniche per calcolare un arco sicuro. Entrambe queste tecniche utilizzano il concetto di taglio di un grafo.

Tagli di un grafo

Dato un grafo non orientato G=(V,E), un taglio è una partizione di V in due sottoinsiemi.

Un arco attraversa il taglio se uno dei suoi estremi è in una partizione, e l'altro nell'altra.

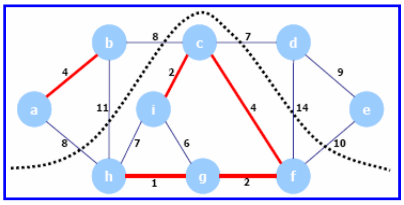
Un taglio rispetta un insieme A di archi se nessun arco di A attraversa il taglio.

Un arco si dice leggero se ha peso minimo tra gli archi che attraversano il taglio.

Esempio:

Siano gli archi in rosso un sottoinsieme **A** di E. Il taglio rispetta chiaramente A.

Se invece **A** è l'insieme degli archi adiacenti ai nodi **d** ed **e**, allora l'arco **(d,c)** è leggero per il taglio dato



Federica 13/12/2007
Facoltà di Scienze Matematiche Fisiche Naturali

Arco sicuro per G

Teorema:

- Sia G=(V,E) un grafo non orientato e connesso con una funzione peso w a valori reali definita su E.
- Sia A un sottoinsieme di E contenuto in un qualche albero di copertura minimo per G.
- Sia (S,V-S) un qualunque taglio che rispetta A.
- Sia (u,v) un arco leggero che attraversa (S,V-S).
- Allora l'arco (u,v) è sicuro per A

In seguito analizziamo in dettaglio due algoritmi che calcolano un BST di un grafo formato da tutti archi sicuri calcolati utilizzando una tecnica greedy:

- **Algoritmo di Kruskal**
- **Algoritmo di Prim**

back next

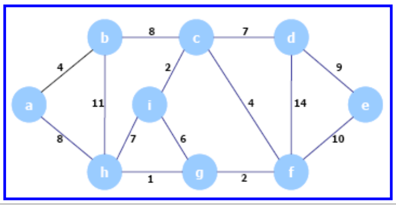
Algoritmo di Kruskal

MST-Kruskal(G,w)

```
A←∅
for ogni vertice v di V
    do Make-Set(V) \\ crea un insieme per ogni vertice V \\
ordina gli archi di E per peso w non decrescente
for ogni arco (u,v) di E, in ordine di peso non decrescente
    do if Find-Set(u)≠Find-Set(v)
        A←A unito a {(u,v)}
           Union(u,v) \\ fusione in un unico insieme degli insiemi di u e v
return A
```

Esempio:

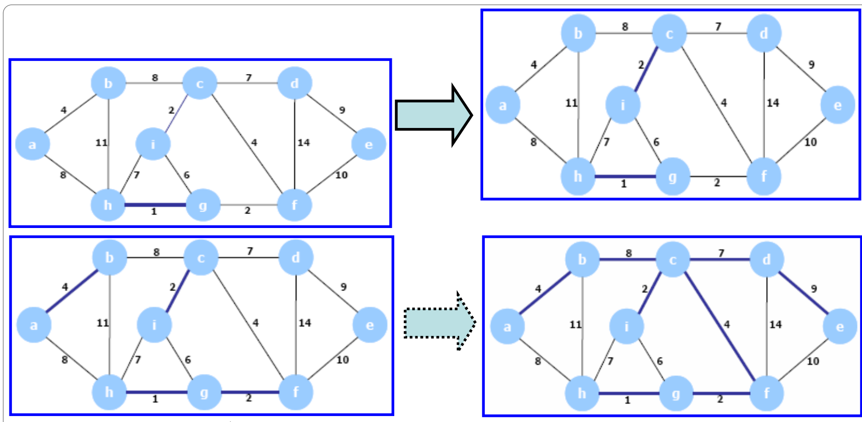
Si consideri il seguente grafo. Inizialmente ogni insieme costituisce un insieme a se.



back next

Federica unina.it 13/12/2007 16 Facoltà di Scienze Matematiche Fisiche Naturali

MST sull'esempio dato



La complessità dell'algoritmo di Kruskal dipende fondamentalmente dall'ordinamento degli archi pesati che prende tempo O(E log E).

back next

Algoritmo di Prim

L'algoritmo di Prim parte da un nodo radice r ed espande ad ogni passo l'albero di copertura minimo A, sino a che questo non copre tutti i vertici.

Ad ogni passo viene aggiunto all'albero un arco leggero che collega un vertice in A ad un vertice in V- A.

Per la scelta dell'arco leggero si usa una coda di priorità.

Ad ogni istante la coda di priorità contiene tutti i vertici non appartenenti all'albero A.

La posizione di ciascun vertice v nella coda dipende dal valore di un campo chiave key[v], corrispondente al minimo tra i pesi degli archi che collegano v ad un qualunque vertice in A.

Se v non è collegato a nessun vertice in A, allora key[v]=∞.

Esempio:

MST-Prim(G,w,r)

```
Q←V[G]
for ogni u di Q
    do key[u]← ∞ \\ chiave massima a tutti i nodi \\
key[r]←0              \\ chiave minima alla radice \\
π[r]←NIL              \\ r non ha padre
while Q≠∅
   do u←Extract-Min(Q)
      for ogni v in Adj[u]
          do if v è in Q e w(u,v)<key[v]
                 π[v]←u
                 key[v]←w(u,v)
```

O(V) Usando buildheap

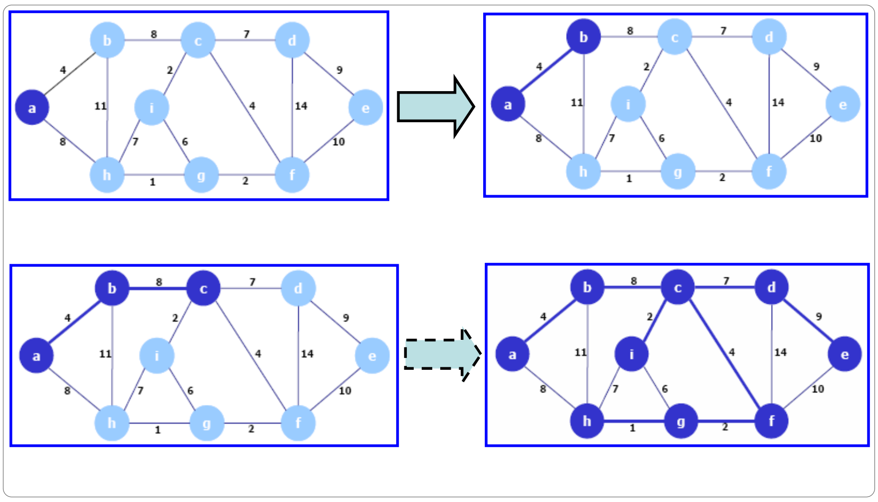
O(log V)

O(E)

O(log V)

Studi approfonditi dimostrano che la complessità asintotica dell'algoritmo di Prim è migliore dell'algoritmo di Kruskal

Esempio



Esercizio

Implementare in linguaggio C
l'algorimo di Prim