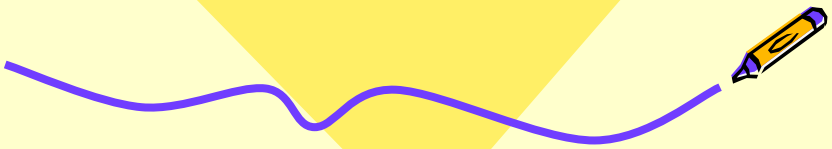
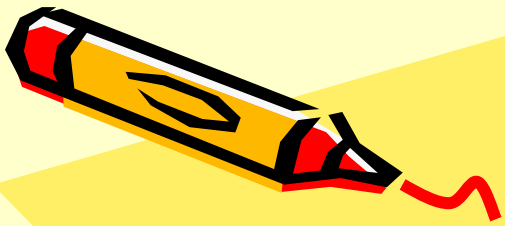
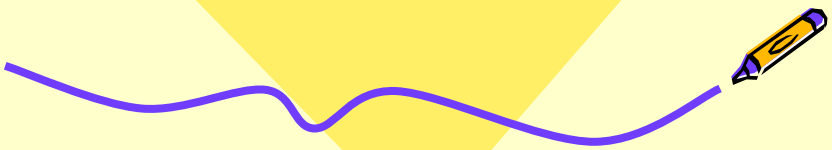
Laboratorio di Algoritmi e Strutture Dati

Aniello Murano
http://people.na.infn.it/~murano/





Grafi pesati e alberi minimi di copertura

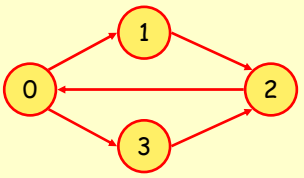


Riepilogo delle lezioni precedenti

- 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
  - Interrogazione di un grafo: calcola grado, stampa grafo, ecc.)
  - Modifica di un grafo: Aggiungi/rimuovi vertice, aggiungi/rimuovi arco, ecc.
  - Visita di tutti i nodi di un grafo (BFS e DFS).
- Oggi vedremo nuovi algoritmi per il calcolo di un grafo trasposto e di una componente fortemente connessa. Inoltre, saranno richiamati i **Grafi Pesati** e verranno introdotti due algoritmi per il calcolo di un albero minimo di copertura

Grafo Trasposto

- Sia G=(V,E) un grafo orientato. Il trasposto di G è un grafo G'=(V,E'), dove E' è l'insieme degli archi (v,u) tali che (u,v) è un arco di E.
- Dunque, il grafo trasposto G' è il grafo G con tutti i suoi archi invertiti.
- Per esempio, si consideri il seguente grafo, la sua rappresentazione e quella del suo trasposto con matrici di adiacenza:



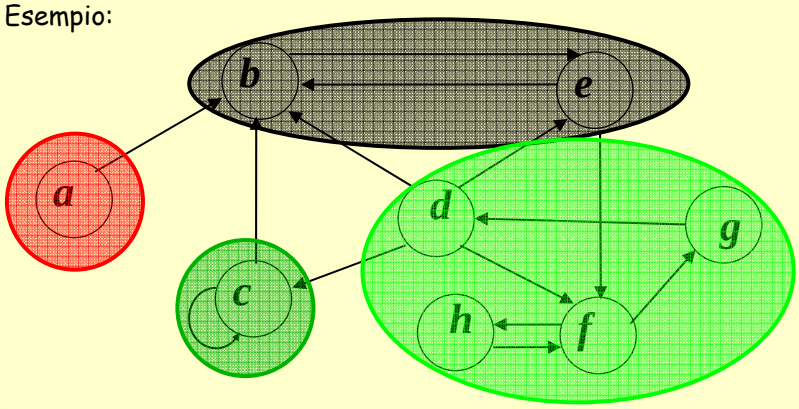
	0	1	2	3
0	0	1	0	1
1	0	0	1	0
2	1	0	0	0
3	0	0	1	0

	0	1	2	3
0	0	0	1	0
1	1	0	0	0
2	0	1	0	1
3	1	0	0	0

- Nel caso di rappresentazione con matrice di adiacenza, il grafo G' è rappresentato dalla trasposta di G che è dunque calcolata in O(V).
- Esercizio: Implementare un algoritmo efficiente per la computazione del trasposto di un grafo rappresentato con liste di adiacenza e confrontare la complessità asintotica dell'algoritmo con quella dell'algoritmo precedente.

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

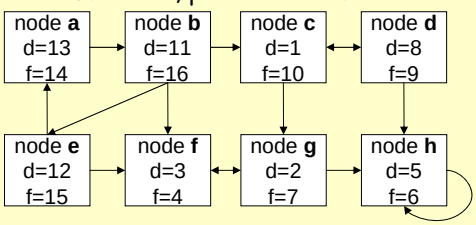
- Dato un grafo, l'operazione di decomposizione nelle sue componenti fortemente connesse ha molte applicazioni pratiche
- Per esempio:
  - la disposizione di sensi unici in una città,
  - I vincoli tra variabili all'interno di un programma, che devono dunque essere presi in considerazione contemporaneamente,
  - la correlazione di moduli di un programma e dunque la capacità per un compilatore di raggruppare moduli in modo efficiente

Algoritmo per SCC

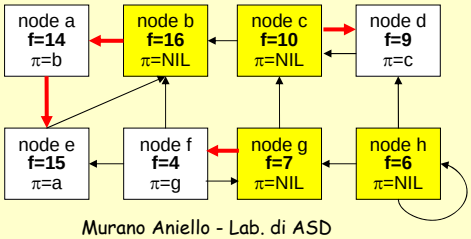
- L'algoritmo che proponiamo per la decomposizione di un grafo nelle sue componenti fortemente connesse usa due visite in profondità, una sul grafo G e l'altra sul grafo trasposto G':
- Algoritmo per SCC:
  - Chiamare DFS(G) per computare f[v] (l'ordine finale di visita) per ogni vertice v
  - Calcolare il grafo trasposto G' di G
  - Chiamare DFS(G'), ma nel ciclo principale della DFS e si considerino i vertici in ordine decrescente di f[v]
  - Restituire i vertici di ciascun albero calcolato con la DFS(G') come una SCC
- Per calcolare f[v], si noti che nella vista in profondità, a ciascun vertice si possono associare due tempi: il primo corrisponde a quando il nodo è scoperto, il secondo corrisponde a quando la ricerca finisce di esaminare i suoi adiacenti.

Esempio

DFS su G, partendo da c.

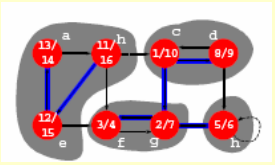


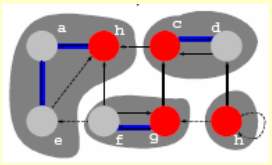
DFS su G' (il grafo trasposto di G) partendo da g (con f massima).

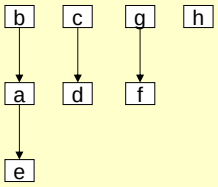


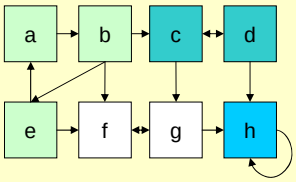
Strongly-Connected Components

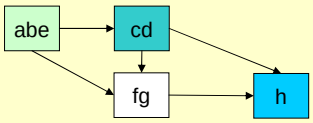
- Di seguito riportiamo i 4 alberi risultanti che formano i quattro SCC del grafo dato.









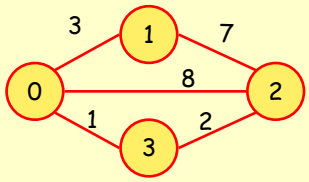


Grafi pesati

- La struttura di un grafo può essere generalizzata associando un valore numerico, detto *peso*, ai suoi archi. Un grafo di tale genere si dice *pesato*. I grafi pesati vengono rappresentati mediante
  - Matrici di adiacenza: se l'elemento di indici *i, j* della matrice di adiacenza è un valore diverso da 0 esso è il peso dell'arco *(i,j)*, altrimenti non esiste un arco fra i nodi *i* e *j*;
  - Liste di adiacenza: un elemento della lista di adiacenza al nodo *i* contiene un campo per memorizzare il nome del nodo adiacente (ad esempio, *j*), un campo per memorizzare il peso dell'arco (nell'esempio, il peso dell'arco *(i,j)*) ed un campo per memorizzare il puntatore all'elemento successivo nella lista.

Rappresentazione con matrice

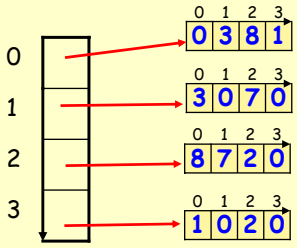
- Per esempio, si consideri il seguente grafo:



Matrice di adiacenza

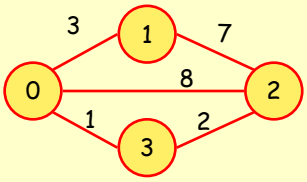
	0	1	2	3
0	0	3	8	1
1	3	0	7	0
2	8	7	2	0
3	1	0	2	0

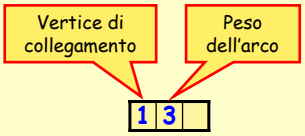
Matrice come vettore di puntatori

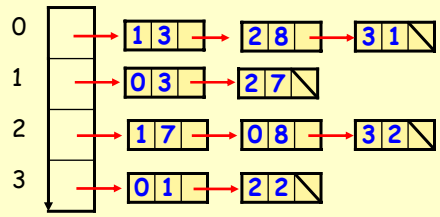


Rappresentazione con liste

- Si consideri di nuovo il grafo dell'esempio precedente:







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
```

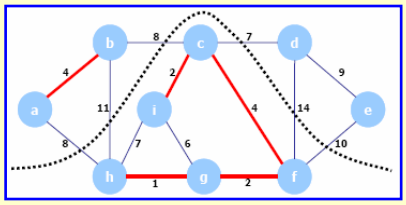
- Si definisce sicuro un arco che aggiunto ad un sottoinsieme di un albero di copertura minimo produce ancora un sottoinsieme di un albero di copertura minimo.
- In seguito vedremo due tecniche per calcolare un arco sicuro. Entrambe queste tecniche utilizzano il concetto di **taglio** di un grafo.

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



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

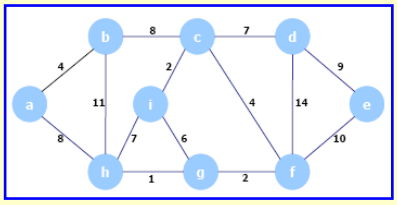
- In seguito analizziamo in dettaglio due algoritmi che calcolano un BST di un grafo formato da tutti archi sicuri calcolati utilizzando una tecnica greedy:
  - **Algoritmo di Kruskal**
  - **Algoritmo di Prim**

Algoritmo di Kruskal

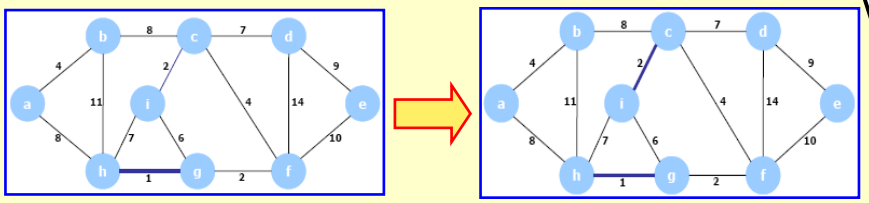
- MST-Kruskal(G,w)
  1. A←∅
  2. **for** ogni vertice v di V
  3. **do** Make-Set(V) \\ crea un insieme per ogni vertice V \\
  4. ordina gli archi di E per peso w non decrescente
  5. **for** ogni arco (u,v) di E, in ordine di peso non decrescente
  6. **do if** Find-Set(u)≠Find-Set(v)
  7. A←A unito a {(u,v)}
  8. Union(u,v) \\ fusione in un unico insieme degli insiemi di u e v
  9. **return** A

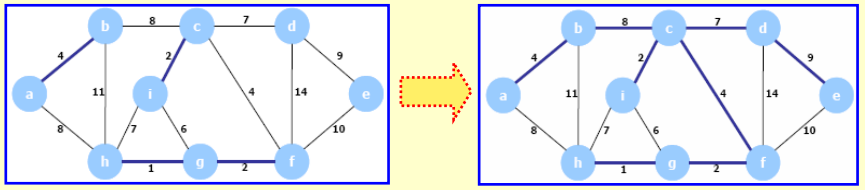
- Esempio:

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



MST sull'esempio dato





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

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)

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

O(V) Usando buildheap

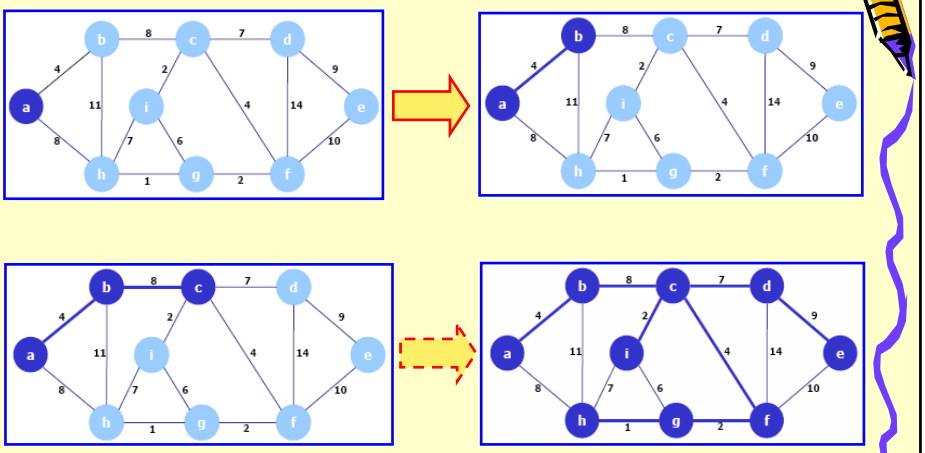
O(log V)

O(E)

O(log V)

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

Esempio



Progetto di Laboratorio

- Problema del venditore:
- Si supponga di avere un insieme di città e di conoscere in che modo sono collegate e la lunghezza di ogni singolo collegamento.
- Si supponga inoltre che un venditore voglia fare affari in queste città e conosca per ogni singola città alcuni parametri quali somme di denaro disponibili e tempi obbligatori di sosta in ogni singola città per ottenere tali soldi.
- Si gestiscano utilizzando una struttura dati opportuna le seguenti operazioni discutendone le complessità asintotiche:
  - Inserimento/Eliminazione di una città.
  - Inserimento/Eliminazione di collegamento tra città.
  - Stampa di una possibile visita di tutte le città.