Minimo Albero Ricoprente

Il problema della definizione di un Minimo Albero Ricoprente trova applicazione pratica in diverse aree di studio, quali ad esempio la progettazione di circuiti elettronici e la progettazione di reti di comunicazione.

Come vedremo, il problema che ci accingiamo ad esaminare è risolvibile mediante l'applicazione di un approccio che effettua di volta in volta la scelta ritenuta localmente migliore: l'approccio Greedy. Rappresentano una applicazione di tale strategia di risoluzione l'algoritmo di Kruskal, che verrà presentato all'interno della nostra esposizione.

Introduciamo pertanto il problema del Minimo Albero Ricoprente partendo da un caso di studio: la progettazione di una rete di comunicazione a costo minimo di realizzazione.

1. Introduzione

Supponiamo di dover risolvere il seguente problema.

Problema. *Sia S={A, B, C, D} un insieme di siti geografici dove ogni sito, ad esempio A, può essere messo in comunicazione con un qualsiasi altro sito, ad esempio B, attraverso un canale di comunicazione diretta, A→B, o attraverso un canale di comunicazione indiretta che attraversa uno o più siti, ad esempio A→C→D→B.*
Si consideri inoltre la seguente tabella riportante i costi di realizzazione dei canali di comunicazione diretta:

	A	**B**	**C**	**D**
A		4	3	1
B	4		5	4
C	3	5		6
D	1	4	6	

Progettare una rete avente costo minimo di realizzazione e che renda possibile la comunicazione, diretta o indiretta, tra tutti i siti geografici.

Analogamente alla risoluzione di un qualsiasi altro problema, il passo propedeutico alla risoluzione è rappresentato dalla corretta formalizzazione di tutti gli elementi. Procediamo quindi in tal senso ed introduciamo un concetto utile per la rappresentazione dei siti e dei costi dei canali di comunicazione diretta.

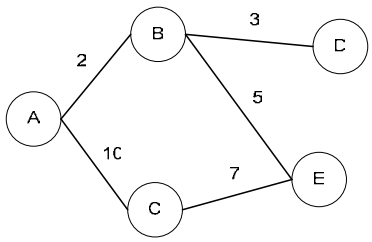
Definizione. Sia G = (V, E) un grafo, si dice che esso è **pesato** se sull'insieme degli archi E di G è definita una funzione, ω, tale che:

$$\omega: e \in E \rightarrow \omega(e) \in \Re$$

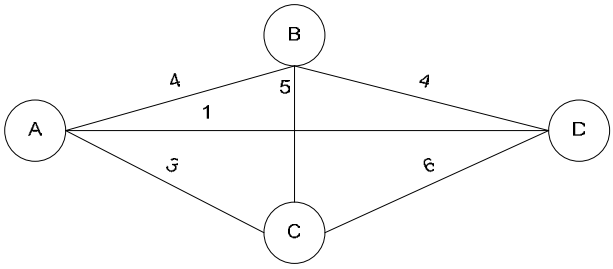
Un grafo è quindi pesato se ad ogni arco è associato un valore numerico. La funzione ω è chiamata ***funzione di peso*** ed i valori da essa assunti sono chiamati ***pesi degli archi***.

Esempio

Segue la raffigurazione di un grafo pesato.



Il grafo pesato è uno strumento valido per la rappresentazione dei siti e dei costi di comunicazione diretta introdotti nel nostro problema. Infatti, se associamo ad ogni sito geografico un vertice del grafo e rappresentiamo sia i canali di comunicazione diretta che i costi ad essi associati come archi pesati del grafo, possiamo rappresentare il problema introdotto mediante la seguente raffigurazione:



Scelta la giusta rappresentazione vediamo ora come caratterizzare la soluzione del nostro problema. Allarghiamo ulteriormente il campo delle nostre conoscenze e introduciamo le seguenti definizioni.

Definizione. Sia $G = (V, E)$ un grafo non orientato e connesso. Si definisce albero ricoprente di G un sottografo $T \subseteq G$ tale che:

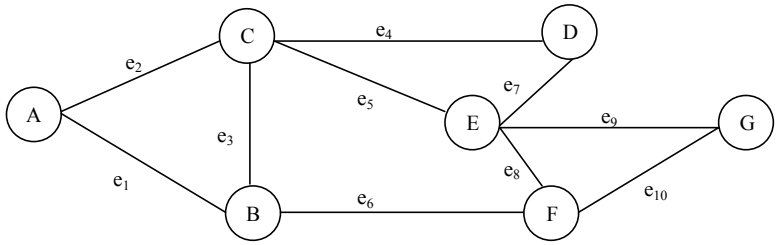
a) T è un albero;

b) T contiene tutti i vertici di G.

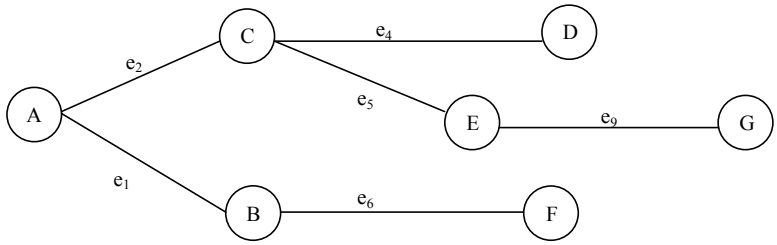
Esempio

Sia G = (V, E) il seguente grafo, dove:

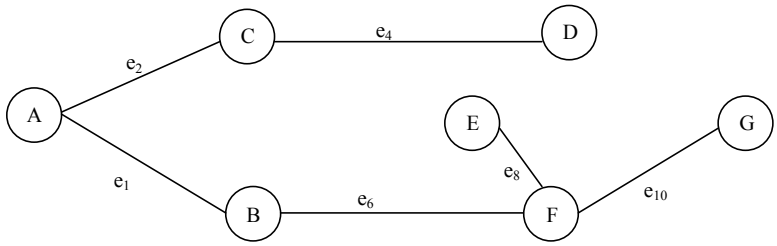
- *V={ A, B, C, D, E, F, G}*
- *E={ e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10} }*



Un albero ricoprente di G è:



Così come un altro albero ricoprente di G è:



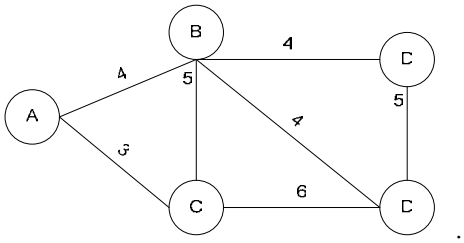
Se il grafo oltre ad essere connesso e non orientato possiede anche la particolarità di essere pesato, possiamo dar vita anche al concetto di costo dell'albero ricoprente.

Definizione. Sia $G = (V, E)$ un grafo non orientato, connesso e pesato. Sia inoltre $T \subseteq G$ un albero ricoprente di G, si definisce costo dell'albero ricoprente di T, $\omega(T)$, la somma dei costi degli archi contenuti in T, ossia:

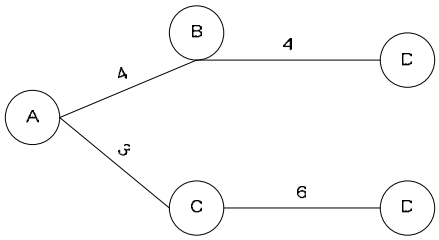
$$\omega(T) = \sum_{e \in T} \omega(e)$$

Esempio

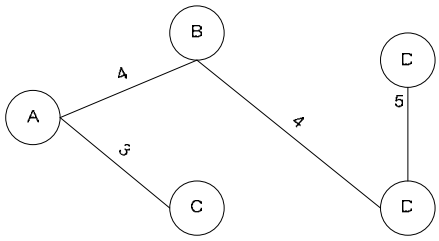
Sia considerato il seguente grafo pesato



Il costo del seguente albero ricoprente è dato dalla somma dei costi degli archi: 17.

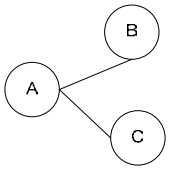


Il costo del seguente albero ricoprente è invece uguale a 16.

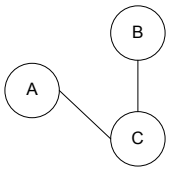


Consideriamo nuovamente il problema introdotto inizialmente ed osserviamo una particolare proprietà della soluzione.

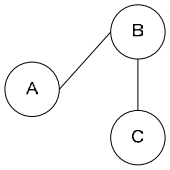
Siano considerati solamente tre siti qualsiasi ad esempio A, B, C. La rete di costo minimo che dovrà permettere la comunicazione tra i tre nodi dovrà ad ogni modo avere una struttura del tipo:



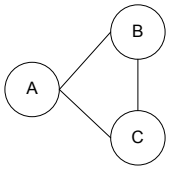
oppure del tipo:



o infine del tipo:



sicuramente la soluzione non potrà essere costituita da una rete avente una struttura del tipo:



poichè in tal caso daremo vita ad un collegamento ridondante che incide comunque sui costi. In sostanza, la soluzione del nostro problema non dovrà in alcun modo contenere cicli.

Detto questo, se proiettiamo tale considerazione verso l'intera struttura grafo rappresentante il problema, se consideriamo che tutti i siti devono essere messi in comunicazione tra loro ed infine se consideriamo che il costo di realizzazione dovrà

essere minimo possiamo asserire che la tipologia di rete soluzione del nostro problema dovrà essere priva di cicli, dovrà contenere tutti i vertici del grafo ed inoltre la somma dei costi di connessione diretta dovrà essere minima. In definitiva la soluzione del problema è rappresentata da un albero ricoprente avente costo minimo, ovvero da un Minimum Spanning Tree (MST).

2 Approccio greedy per la ricerca di un MST

Nella ricerca di un MST adotteremo la tecnica greedy che essenzialmente ci suggerisce di costruire l'MST un arco alla volta effettuando scelte ritenute localmente valide (scelte golose). Vediamo prima le seguenti definizioni.

Definizione. Sia *G=(V, E)* un grafo non orientato e connesso. Si definisce ***taglio*** del grafo *G* una partizione, *(S, V-S)*, dell'insieme dei vertici del grafo *V*.

Esempio

Sia G=(V, E) un grafo non orientato dove V={ 1, 2, 3, 4, 5, 6 }. Rappresentano tagli del grafo le partizioni:

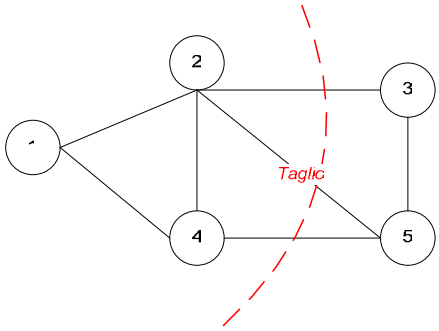
a. *S= {1, 2} e V-S = {3, 4, 5, 6}*
b. *S= {1, 3, 4} e V-S = {2, 5, 6}*
c. *S= {1, 3, 6} e V-S = {2, 4, 5}*

Definizione. Sia *G=(V, E)* un grafo non orientato e connesso e sia *(S, V-S)* un taglio di *G*. Si dice che un arco *(u, v)* $\in E$ del grafo *G* ***attraversa il taglio*** se:

1. $u \in S$ e $v \in V-S$;
2. $u \in V-S$ e $v \in S$.

Esempio

Sia G=(V, E) il grafo in figura e sia la coppia (S, V-S) dove S={1, 2, 4} e V-S={3, 5} un taglio di G.



Gli archi (2,3), (2,5) e (4,5) attraversano il taglio definito su G. Gli archi (3,5), (2,4), (1,2), e (1,4) non attraversano il taglio.

Definizione. Siano *G=(V, E)* un grafo non orientato e connesso, *(S- V-S)* in taglio di *G* ed *A⊆E* un insieme di archi di *G*. Si dice che il taglio *(S, V-S)* ***rispetta*** l'insieme *A* se nessun arco di *A* attraversa il taglio.

Esempio

Riallacciandoci all'esempio precedente abbiamo che il taglio S={1, 2, 4} e V-S={3, 5} definito su G:

- *rispetta l'insieme A se A = { (1,2); (1,4) };*
- *rispetta l'insieme A se A = {(1,2); (1,4); (2,4)};*
- *non rispetta l'insieme A se A = {(1,2); (2,3); (2,4)}, poichè (2,3) attraversa il taglio;*
- *non rispetta l'insieme A se A = {(1,4); (2,5); (4,5)}, poichè (2,5) e (4,5) attraversano il taglio;*

Torniamo all'approccio Greedy. Nella risoluzione del problema del MST l'approccio greedy consiste nel costruire, a partire da un singolo nodo, l'albero di costo minimo arco per arco, effettuando di volta in volta scelte ritenute localmente valide.

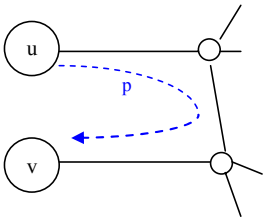
Pertanto, quale è la scelta che di volta in volta bisogna ritenere valida? La risposta a tale domanda la troviamo nell'enunciato del seguente teorema.

Teorema. *Sia G=(V,E) un grafo connesso, non orientato e pesato. Siano inoltre:*

1. *$A \subseteq E$ un sottoinsieme di archi di un MST, che indichiamo con T, di G;*
2. *(S, V-S) un taglio definito su G che rispetta A;*
3. *$(u, v) \in E$ un arco del grafo di peso minimo che attraversa il taglio definito su G*

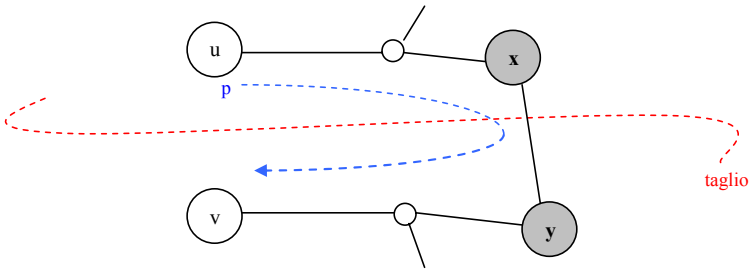
*allora (u,v) è un arco **sicuro** per A, cioè se ad A aggiungiamo anche l'arco (u,v) allora A continua a rappresentare un sottoinsieme di archi di un MST.*

Dimostrazione. Sia *T* un MST che contiene gli archi di *A*. Se *(u,v)* è contenuto in *T* allora il teorema è provato. Assumiamo quindi che *T* non contiente l'arco *(u,v)* e consideriamo il cammino del grafo, *p*, contenuto in *T* che collega i nodi *u* e *v*.



Abbiamo che:

1. il cammino *p* è l'unico cammino di *T* che connette i due nodi, altrimenti *T* sarebbe ciclico;
2. poichè l'arco *(u,v)* attraversa il taglio allora deve esistere almeno un arco del cammino, che indichiamo con *(x,y),* che similmente ad *(u,v)* attraversa il taglio definito su *G.*



Dalla prima proprietà si evince che $T_1 = T-\{(x, y)\} \cup \{(u, v)\}$ è ancora uno spanning tree poichè è aciclico e copre tutti i vertici del grafo. Dimostriamo che T_1 è di costo minimo.
Per ipotesi l'arco *(u, v)* è un arco di costo minimo che attraversa il taglio; per cui $\omega(u, v) \leq \omega(x, y)$. Da questa assunzione si ha:

$$\omega(T_1) = \omega(T) - \omega(x, y) + \omega(u, v) \leq \omega(T)$$

Pertanto, T_1 è un MST cioè uno spaning tree di costo minimo.
Resta quindi da provare che $A \cup (u, v)$ è un sottoinsiemi di archi di T_1.
Per ipotesi *A* è un sottoinsieme di archi di *T* che non contiene né l'arco *(x, y)* e né l'arco *(u,v)* poichè il taglio definito su *G* rispetta *A*. Ora, essendo T_1 uno spanning tree contenente tutti

gli archi di T tranne che l'arco (x, y) abbiamo che se A è contenuto in T allora A è anche contenuto in T_1 ed a maggior ragione $A \cup \{(u, v)\}$ è un sottoinsiemi di archi di T_1. .

Riportiamo l'algoritmo greedy per la ricerca di un MST . L'insieme di archi indicato con A nel teorema appena dimostrato verrà rappresentato mediante l'identificativo SOL.

Algoritmo MST Generico

MST-Greedy

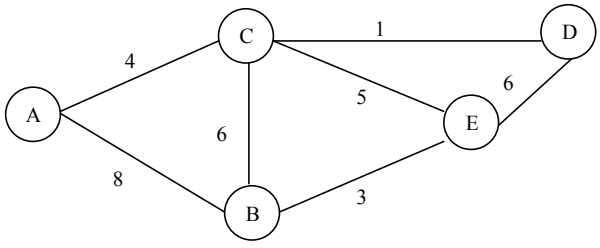
```
{
        GrafoPesato     G
        ListaArchi      SOL
        ListaVertici    S

        inserisci in S un qualsiasi vertice del grafo
        fin quando( S è diverso da V)
        {
                individua l'arco di peso minimo (u,v) che attraversa il taglio (S, V-S)
                inserisci (u,v) in SOL
                inserisci il vertice v in S
        }
}
```

scelta locale

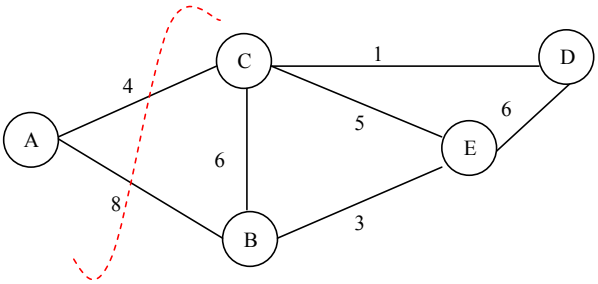
Esempio

In considerazione del grafo G=(V, E) sotto raffigurato, applichiamo l'algoritmo generico per la ricerca di un MST.



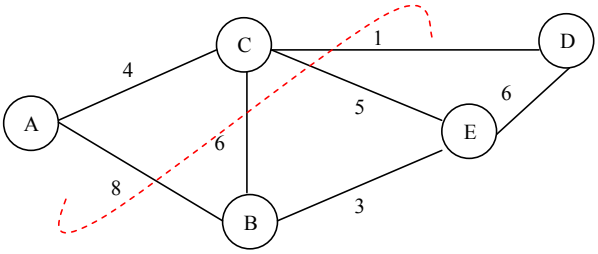
Poniamo inizialmente S={A}

Passo 1
L'arco di costo minimo che attraversa il taglio (S, V-S) è (A,C).



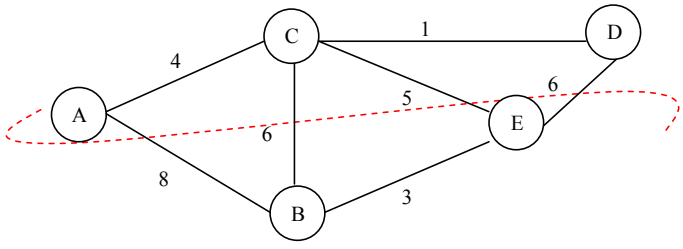
Per cui:
S={ A, C}
SOL = {(A,C)}

Passo 2
L'arco di costo minimo che attraversa il taglio (S, V-S) è (C,D).



Per cui:
S={ A, C, D}
SOL = {(A,C), (C,D)}

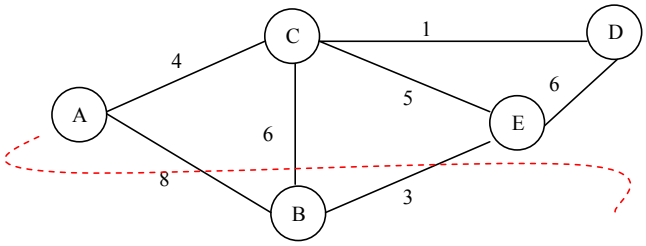
Passo 3
L'arco di costo minimo che attraversa il taglio (S, V-S) è (C,E).



Per cui:
S={ A, C, D, E}
SOL = {(A,C), (C,D), (C,E)}

Passo 4

L'arco di costo minimo che attraversa il taglio (S, V-S) è (E,B).



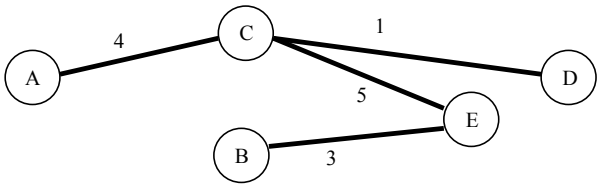
Per cui:

S={ A, C, D, E, B}

SOL = {(A,C), (C,D), (C,E), (B,E)}

Fine algoritmo

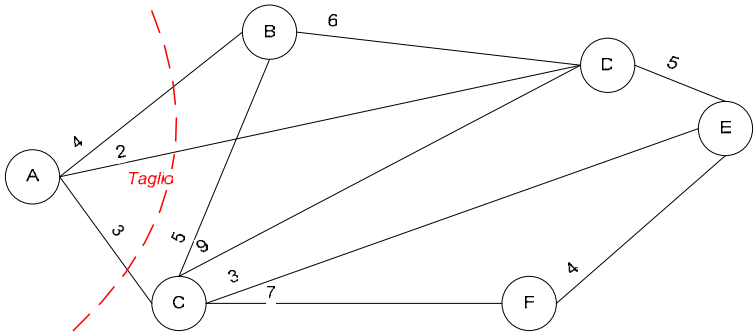
L'insieme S è uguale all'insieme dei vertici del grafo V. L'MST è formato dagli archi contenuti in SOL.



Esempio

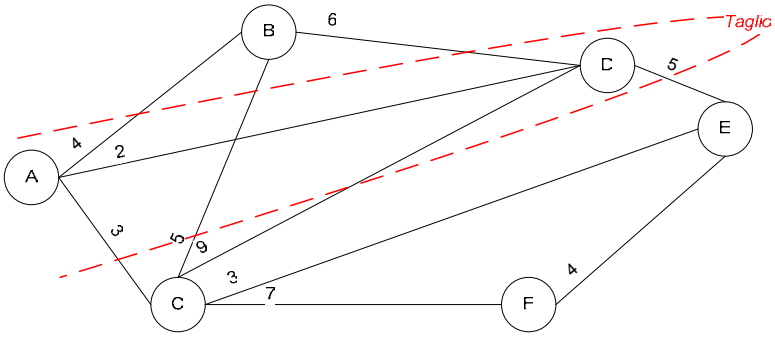
Sia G=(V, E) il grafo sotto raffigurato, applichiamo l'algoritmo generico per la ricerca di un MST.

Poniamo inizialmente S={A}



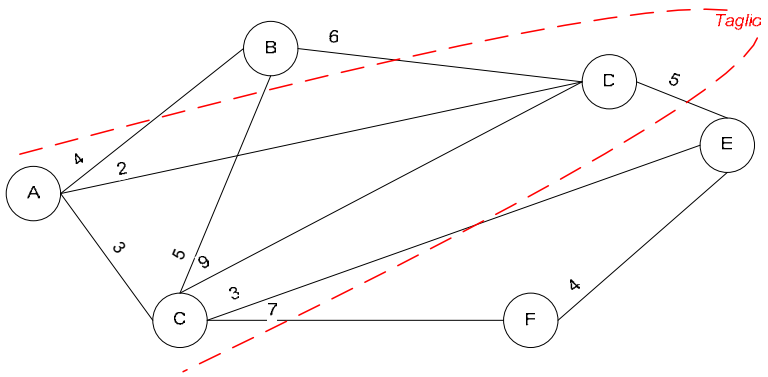
Passo 1

L'arco di costo minimo che attraversa il taglio è(A, D) .
Per cui: S={ A, D} e SOL = {(A,D)}



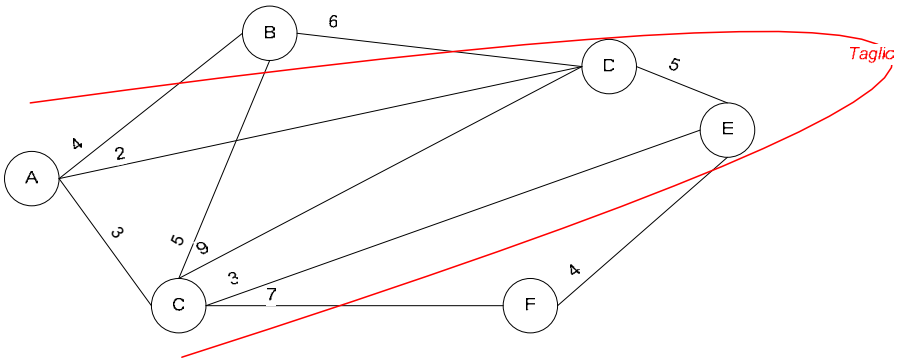
Passo 2

L'arco di costo minimo che attraversa il taglio è(A, C) .
Per cui: S={ A, D, C} e SOL = {(A,D), (A, C)}



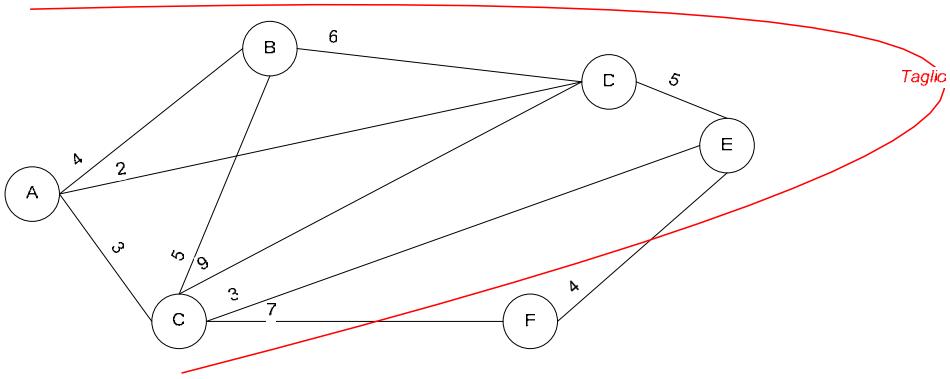
Passo 3

L'arco di costo minimo che attraversa il taglio è(C, E) .
Per cui: S={ A, D, C, E} e SOL = {(A,D), (A, C), (C, E)}



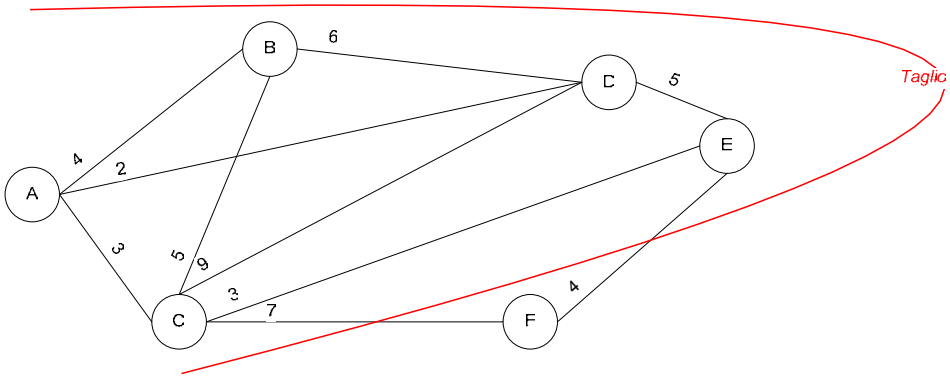
Passo 4

Esistono due archi aventi costo uguale a 4 che attraversano il taglio. Scegliamo di inserire l'arco il vertice B all'interno del taglio.
Per cui: S={ A, D, C, E, B} e SOL = {(A,D), (A, C), (C, E), (A, B)}



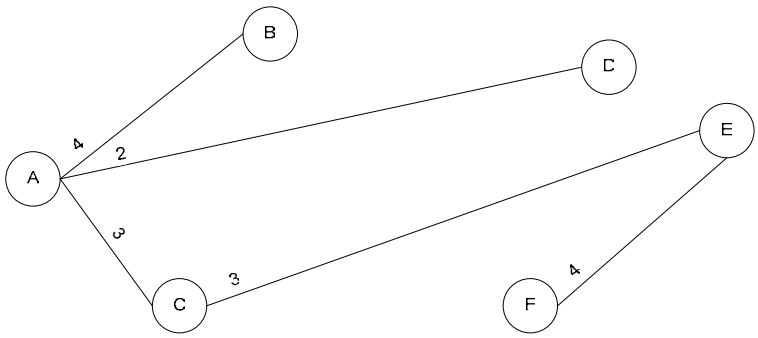
Passo 5

L'arco di costo minimo che attraversa il taglio è(F, E) .
Per cui: S={ A, D, C, E, B, F} e SOL = {(A,D), (A, C), (C, E), (A, B), (F, E)}



Fine Algoritmo

L'insieme S contiene tutti i vertici del grafo. L'MST è formato dagli archi contenuti in SOL..



3. ALGORITMO DI KRUSKAL

L'algoritmo esposto nel precedente capitolo formalizza l'idea "*greedy*" da adottare nella risoluzione del problema del MST. A partire da tale idea, Kruskal ha progettato un algoritmo che applica i concetti esposti e fornisce una soluzione pratica realizzabile attraverso l'impiego dell'ADT Union Find.

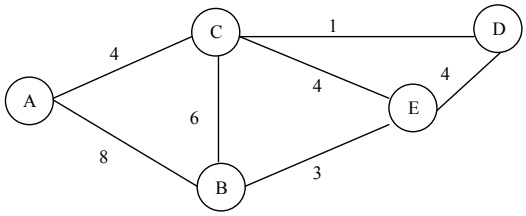
L'idea introdotta da Kruskal consiste nell'isolare inizialmente tutti i vertici del grafo e definire quindi una foresta di alberi disgiunti formati da un singolo vertice. A partire da tale condizione Kruskal suggerisce di considerare, iterativamente, tutti gli archi del grafo in ordine di peso non decrescente e di applicare ad ogni iterazione la seguente regola:

> *se i vertici dell'arco non appartengono allo stesso albero allora unisci i due alberi altrimenti ignora l'arco.*

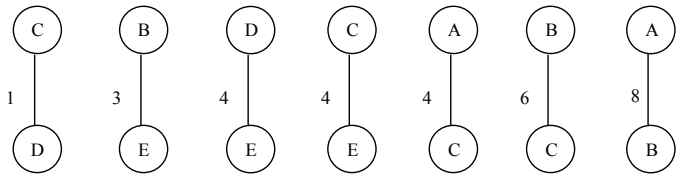
L'algoritmo dovrà terminare quando la foresta sarà costituita da un singolo albero.

Esempio

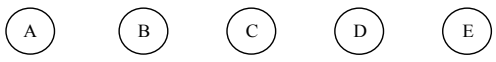
Sia G=(V, E) il grafo sotto raffigurato.



Allineiamo gli archi di G in ordine non decrescente

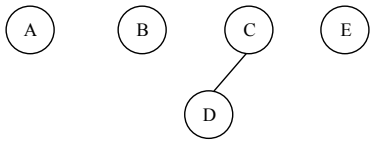


e definiamo la seguente foresta di alberi, dove ogni albero è costituito da un solo nodo.

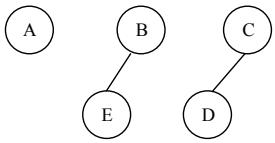


Applichiamo quindi le iterazioni.

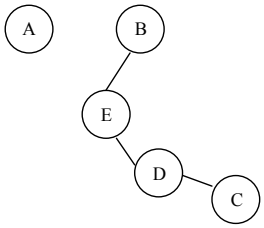
Passo 1
Arco da considerare: (C, D)
Azione: poichè i vertici C e D appartengono ad alberi diversi uniamo i due alberi.
Foresta:



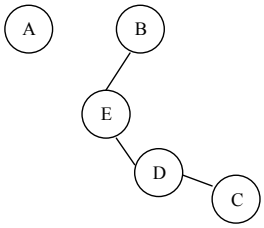
Passo 2
Arco da considerare: (B, E)
Azione: poichè i vertici B ed E appartengono ad alberi diversi uniamo i due alberi.
Foresta:



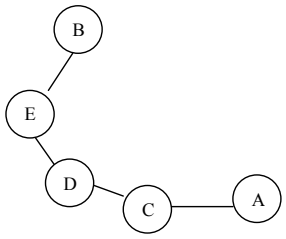
Passo 3
Arco da considerare: (D, E)
Azione: poichè i vertici D ed E appartengono ad alberi diversi uniamo i due alberi.
Foresta:



Passo 4
Arco da considerare: (C, E)
Azione: poichè i vertici C ed E appartengono allo stesso albero l'arco viene ignorato.
Foresta:



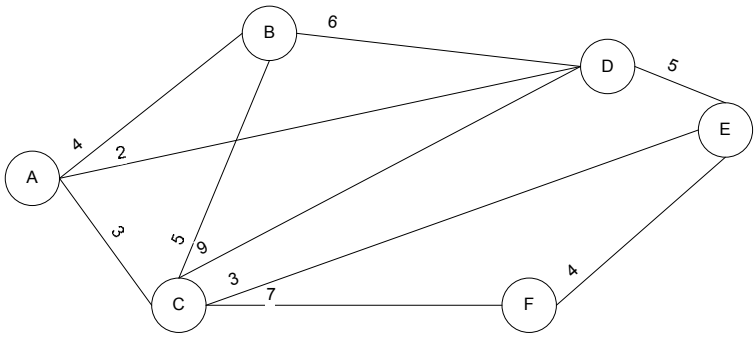
Passo 5
Arco da considerare: (A, C)
Azione: poichè i vertici A e C appartengono ad alberi diversi si uniscono i dei due alberi.
Foresta:



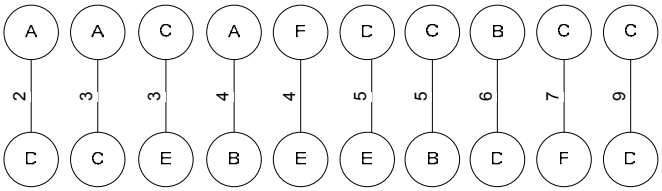
Fine Algoritmo
La foresta contiene un solo albero che rappresenta il minimo albero ricoprente

Esempio

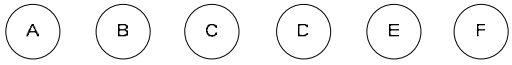
Sia G=(V, E) il grafo sotto raffigurato.



Allineiamo gli archi in ordine non decrescente.

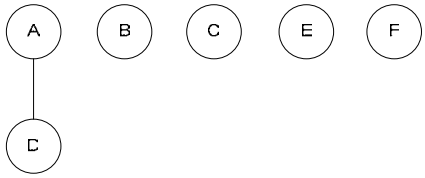


Definiamo la foresta di alberi

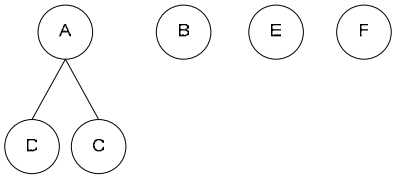


Applichiamo le iterazioni

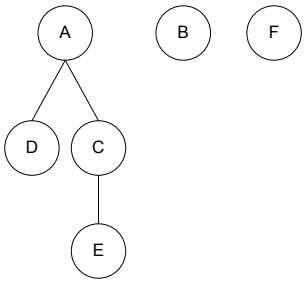
Passo 1
Arco da considerare: (A, D)
Azione: poichè i vertici A e D appartengono ad alberi diversi uniamo i due alberi.
Foresta:



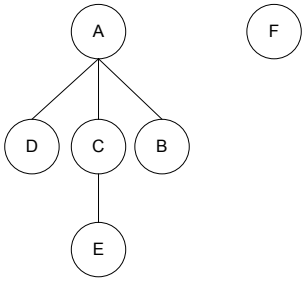
Passo 2
Arco da considerare: (A, C)
Azione: poichè i vertici A e C appartengono ad alberi diversi uniamo i due alberi.
Foresta:



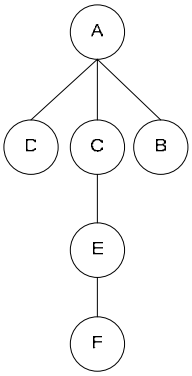
Passo 3
Arco da considerare: (C, E)
Azione: poichè i vertici C e E appartengono ad alberi diversi uniamo i due alberi.
Foresta:



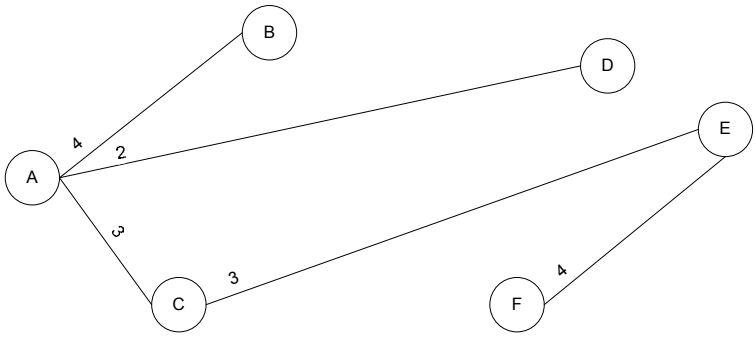
Passo 4
Arco da considerare: (A, B)
Azione: poichè i vertici A e B appartengono ad alberi diversi uniamo i due alberi.
Foresta:



Passo 5
Arco da considerare: (F, E)
Azione: poichè i vertici F e E appartengono ad alberi diversi uniamo i due alberi.
Foresta:



Fine Algoritmo
La foresta contiene un solo albero che rappresenta il minimo albero ricoprente



Dimostriamo ora la correttezza dell'algoritmo.

Teorema. *L'algoritmo di Kruskal calcola correttamente un MST*

Dimostrazione. Notiamo prima che ogni arco dell'unico albero che rimarrà presente all'interno della foresta sarà presentato come l'insieme soluzione.
Sia ora *e* un arco considerato dall'algoritmo ad un generico passo. Si possono avere i due seguenti casi:

a. gli estremi dell'arco appartengono ad alberi uguali. In tal caso l'algoritmo scarta correttamente l'arco poichè se inserissimo l'arco all'interno dell'insieme soluzione introdurremo un ciclo.
b. gli estremi dell'arco appartengono ad alberi diversi. In tal caso *e* attraversa un taglio definibile su G ed essendo *e* il primo arco considerato che collega i due vertici allora *e* è un arco di costo minimo che attraversa tale taglio. Pertanto la scelta di inserire *e* all'interno della soluzione è corretta.

3.1 Implementazione mediante Union-Find

Il concetto alla base dell'algoritmo di Kruskal si evolve in maniera del tutto naturale in una idea implementativa imperniata sull'utilizzo delle strutture dati Union-Find, dove:

- realizziamo, mediante l'attuazione di una sequenza di operazioni makeset, la foresta di alberi contenenti i singoli vertici del grafo
- uniamo gli alberi della foresta mediante l'attuazione dell'operazione di union
- richiamiamo l'operazione di find per valutare se i due estremi dell'arco appartengono allo stesso albero

Algoritmo Kruskal

```
MST-Kruskal
{
    Grafo                G
    Insieme soluzione    A

    per ogni vertice v di G
        Makeset(v)

    ordina gli archi di E in ordine non decrescente di peso
    per ogni arco (u,v)
    {
        se find(u) != find(v) allora
        {
            A ← A∪{(u,v)}
            Union( u, v)
        }
    }

    ritorna A
}
```

L'algoritmo può anche terminare prima di aver analizzato l'intero insieme degli archi. Può rappresentare infatti una circostanza di fine computazione il verificarsi dell'evento *"la foresta contiene un unico albero"*.

Analizziamo ora la complessità sulla base dei concetti introdotti nello studio del problema Union Find.

Teorema. *Sia G=(V, E) un grafo non orientato, connesso e pesato. Siano, inoltre, m ed n rispettivamente il numero di archi ed il numero dei vertici del grafo. La complessità dell'algoritmo di Kruskal nel caso peggiore è O(m log n).*

Dimostrazione. Utilizzando algoritmi efficienti possiamo ordinare gli archi con una complessità pari a *O(m logm)*.
Consideriamo ora il secondo ciclo *for*. Esso viene eseguito al massimo *m* volte. Pertanto, diamo luogo al massimo ad *m* operazioni *find* ed *union*. Applicando la combinazione delle euristiche union by rank e path compression e considerando le *n* operazioni di *makeset* abbiamo che la complessità dell'intera sequenza composta da operazioni di *Makeset, Union* e *Find* è pari a *O((m+n) G(n))*, dove *G* rappresenta la funzione inversa di Ackerman. Essendo ora il numero di archi maggiori o al massimo uguali al numero di vertici meno uno, ossia $m \geq n-1$, possiamo riscrivere tale complessità come *O(m G(n))*.
In considerazione di questi due punti abbiamo che la complessità dell'intero algoritmo è:

$$O(m \log m) + O(m\, G(n)) \qquad \textbf{(1)}$$

Possiamo a questo punto ridurre ulteriormente i termini di *(1)* se osserviamo che il numero di archi non può ad ogni modo superare il numero di nodi al quadrato, ovvero $m<n^2$. Ciò implica infatti che $\log m<\log n^2$, quindi che $\log m < 2 \log n$ ossia che $\log(m) = O(\log n)$.
La *(1)* pertanto può essere riscritta come:

$$O(m \log n) + O(m\, G(n))$$

e poiché la funzione inversa di Ackerman cresce meno velocemente della funzione logaritmo la *(1)* diventa definitivamente uguale a:

$$O(m \log n)$$