Algoritmi e Strutture Dati

Capitolo 12

Minimo albero ricoprente:

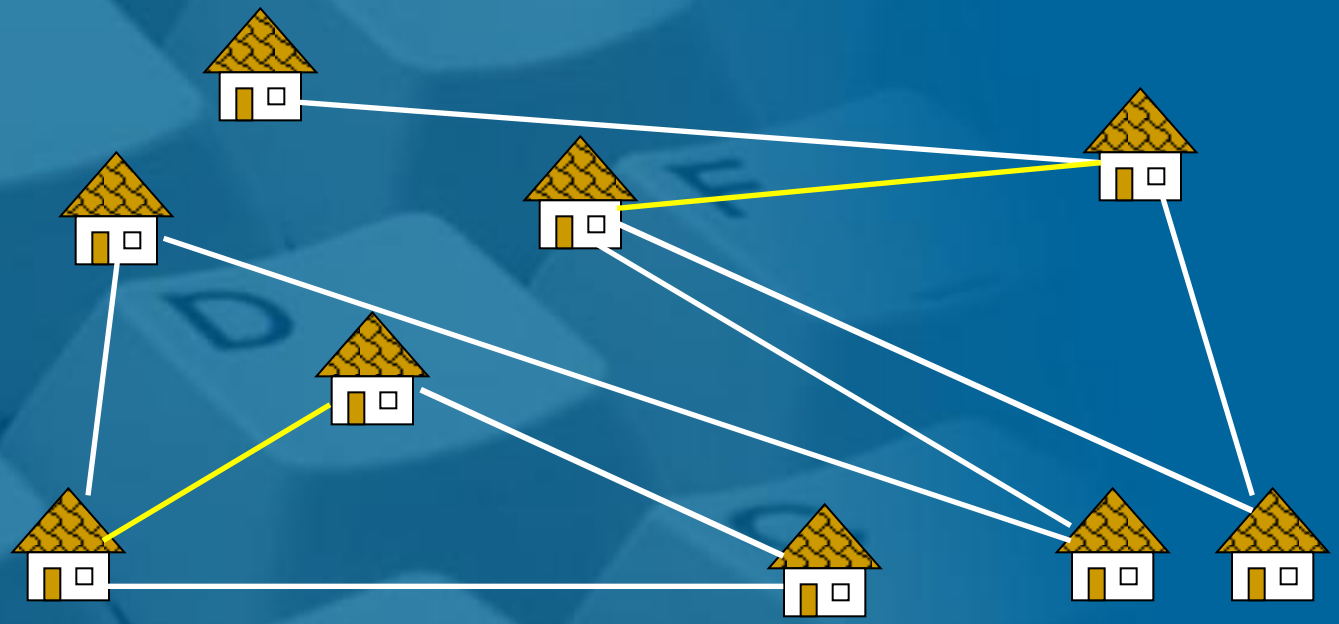
Algoritmo di Kruskal

Progettare una rete stradale

Supponiamo di dover progettare una rete stradale in cui il costo di costruzione di un collegamento tra due abitazioni è direttamente proporzionale alla distanza fisica (euclidea) tra di esse.

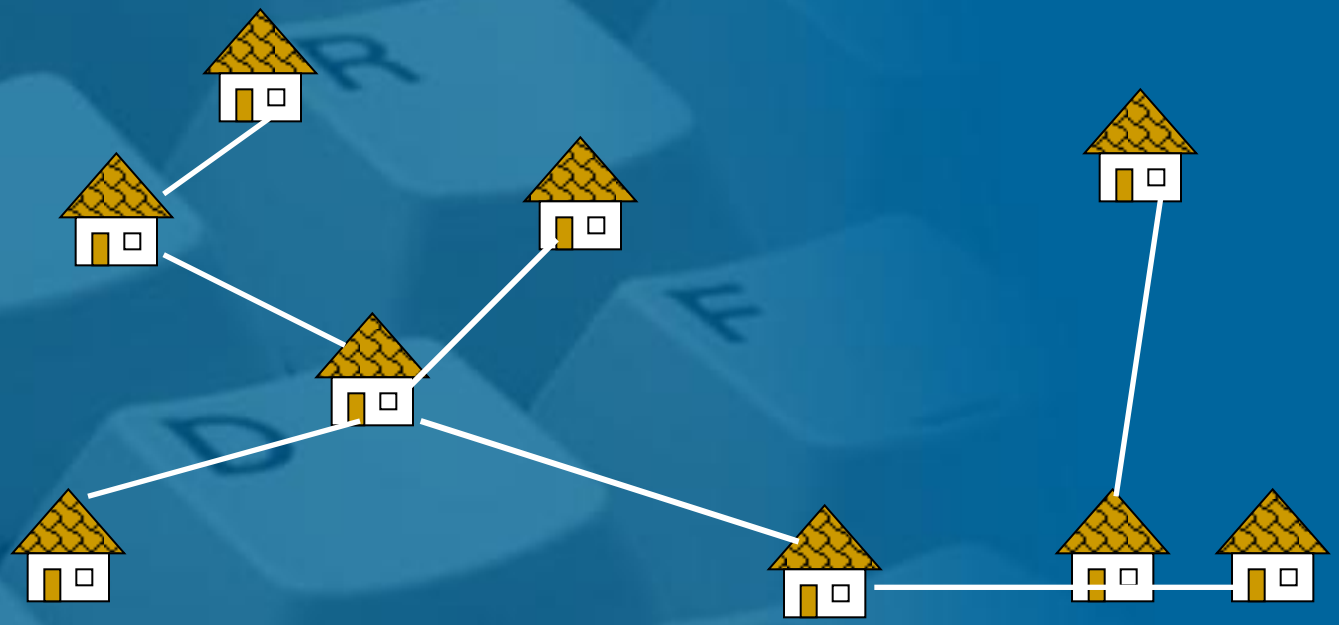
Requisito minimo: connettività tra tutte le abitazioni.

Una soluzione costosa



Usa molti archi, alcuni dei quali sono ridondanti (ovvero, potrebbero essere eliminati senza violare la connettività). Inoltre, ad occhio, gli archi usati sono molto lunghi e quindi costosi.

Una soluzione ottima



Usa il minimo numero di archi (pari al numero di abitazioni meno 1), di lunghezza complessiva minima (fidatevi!). In termini teorici, è un minimo albero ricoprente del grafo completo euclideo avente per nodi le abitazioni, e per pesi degli archi la distanza euclidea tra i relativi estremi.

Definizioni

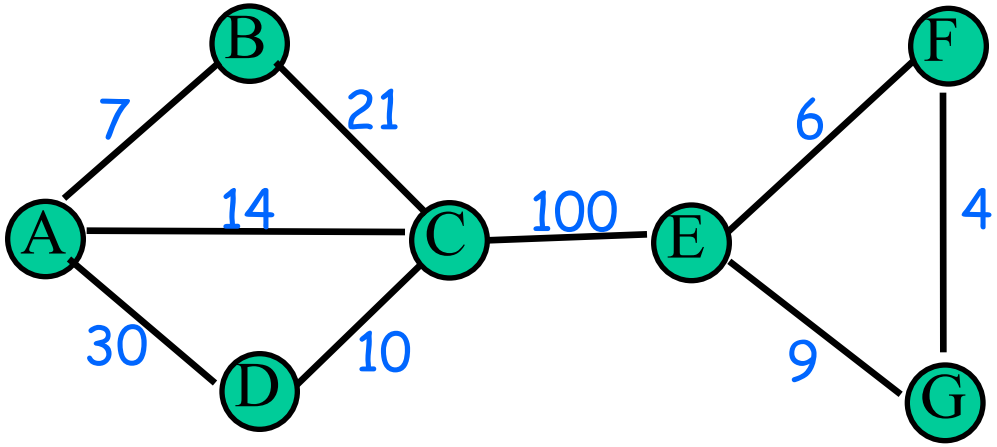
- Sia G=(V,E,w) un grafo non orientato, connesso e pesato (pesi reali). Il peso degli archi rappresenta un generica funzione di costo sugli archi.
- Un albero ricoprente (spanning tree) di G è un sottografo T=(V,E′) di G tale che:
  - T è un albero;
  - T contiene tutti i vertici di G.
- Il costo dell'albero w(T) è la somma dei pesi (costi) degli archi appartenenti all'albero.
- Un minimo albero ricoprente (minimum spanning tree, abbreviato con MST) di G è un albero ricoprente di G avente costo minimo.

Il problema del calcolo di un Minimum Spanning Tree (MST)

- Input:
  - un grafo non orientato e pesato $G=(V,E,w)$
- Soluzione ammissibile:
  - un albero di copertura (uno spanning tree) di G, ovvero un albero $T=(V,F)$ con $F \subseteq E$
- Misura della soluzione (da minimizzare):
  - costo di T: $\sum_{e \in F} w(e)$

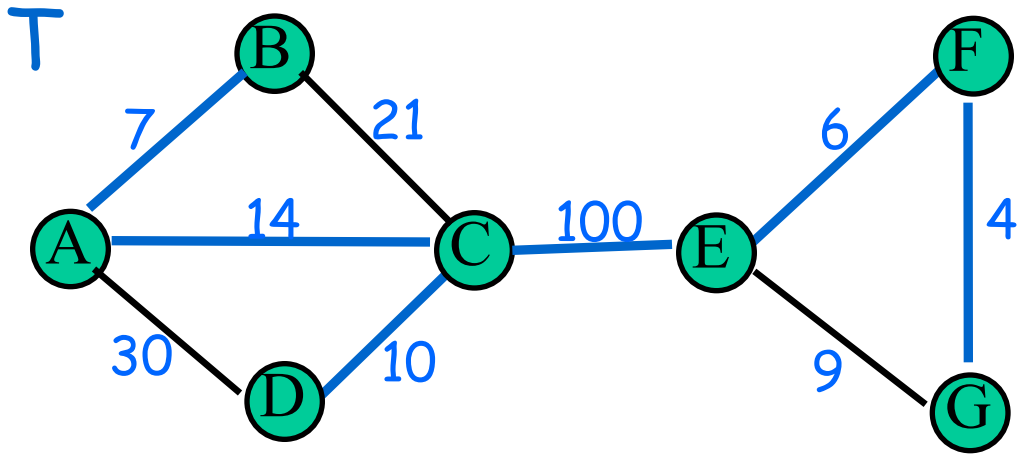
Un esempio

un minimo albero
ricoprente?



Un esempio

un minimo albero
ricoprente?



Esempi

Il minimo albero ricoprente non è necessariamente unico

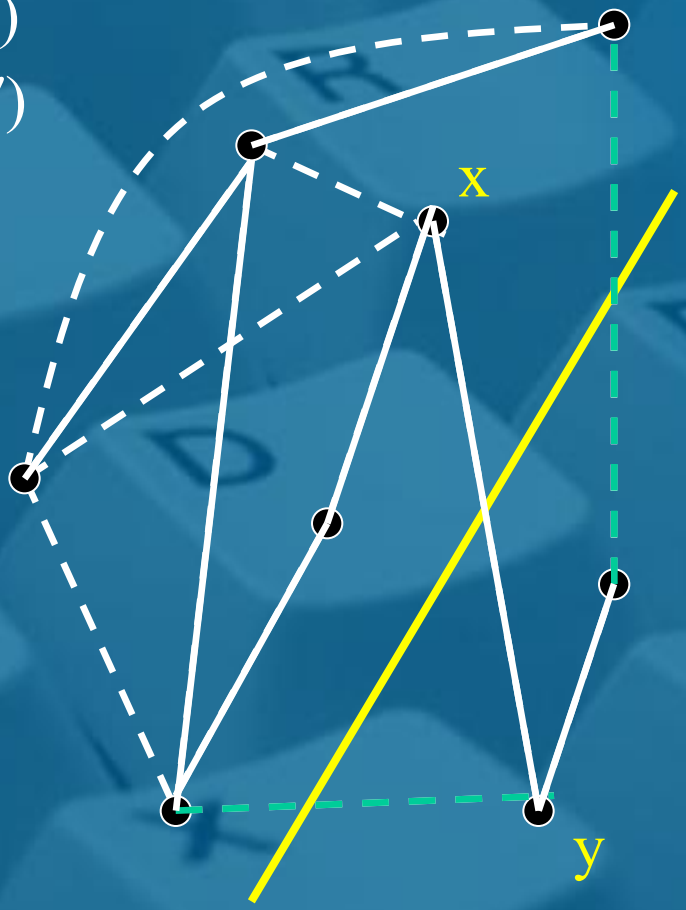
Proprietà dei minimi alberi ricoprenti

Tagli e cicli

- Dato un grafo non orientato G=(V,E), un taglio (X,Y) in G è una partizione dei vertici V in due insiemi (disgiunti): X e Y=V-X.
- Un arco e=(u,v) attraversa il taglio (X,Y) in G se $u \in X$ e $v \in Y$
- Nota: rimuovendo un arco e da un albero T ricoprente G, generiamo un taglio nel grafo G (quello indotto dai due sottoalberi in cui si partiziona T)
- Nota: aggiungendo un arco e=(u,v) ad un albero T, generiamo un ciclo (il cosiddetto ciclo fondamentale di e rispetto a T) costituito da e=(u,v) e dall'unico cammino semplice in T che congiunge u e v

Tagli e cicli: un esempio

G=(V,E)
T=(V,E′)



Rimuovendo l'arco (x,y) da T ottengo un taglio in G, con gli archi blu che attraversano il taglio

Tagli e cicli: un esempio

Aggiungendo l'arco (u,v) a T ottengo un ciclo in T∪ {(u,v)}

Un approccio "goloso"

- Costruiremo un minimo albero ricoprente un arco alla volta, effettuando scelte localmente "golose". Intuitivamente:
  - includeremo nella soluzione archi di costo piccolo che attraversano tagli di G
  - escluderemo dalla soluzione archi di costo elevato che appartengono a cicli in G
- Formalizzeremo il processo come un processo di colorazione degli archi del grafo:
  - archi blu: inclusi nella soluzione
  - archi rossi: esclusi dalla soluzione

Regola del taglio (regola blu)

Scegli un taglio in G che non è attraversato da **archi blu**. Tra tutti gli archi non ancora colorati che attraversano il taglio, scegline uno di costo minimo e coloralo di blu.

Infatti, ogni albero ricoprente G deve contenere almeno un arco che attraversa il taglio (per garantire la connettività), e dimostreremo che è corretto scegliere quello di costo minimo.

Regola del ciclo (regola rossa)

Scegli un ciclo in G che non contiene **archi rossi**.
Tra tutti gli archi non ancora colorati del ciclo, scegline uno di costo massimo e coloralo rosso.

Infatti, ogni albero ricoprente G deve escludere almeno un arco del ciclo (per garantire l'aciclicità), e dimostreremo che è corretto eliminare quello di costo massimo.

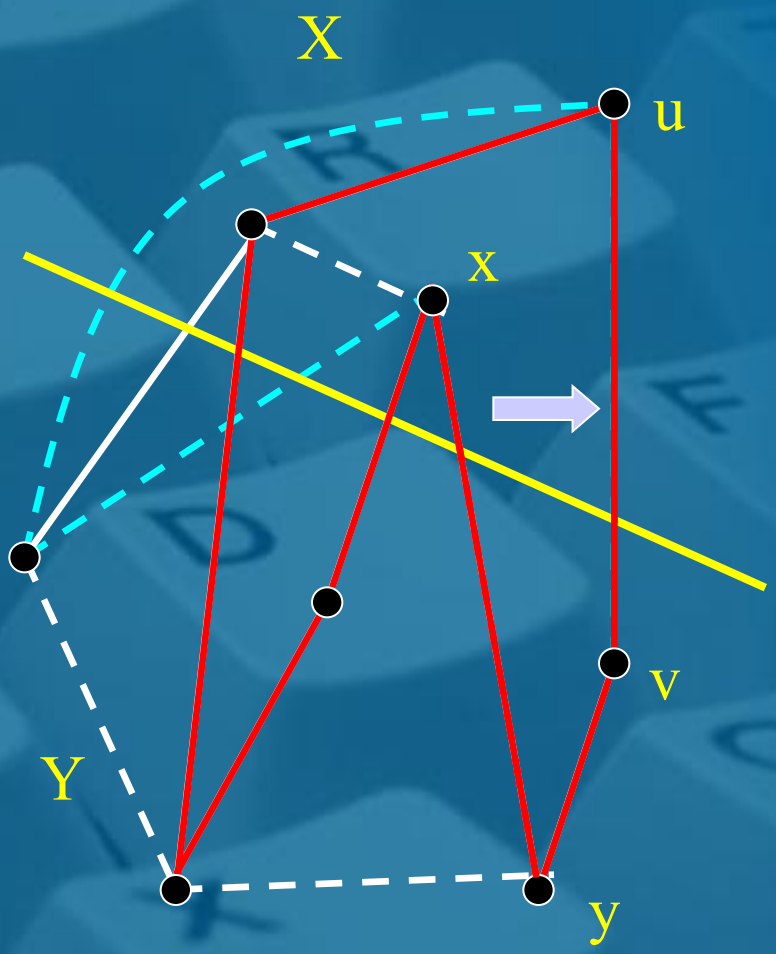
L'approccio "goloso"

- L'approccio goloso applica una delle due regole ad ogni passo, finché tutti gli archi sono colorati
- Dimostreremo che ad ogni passo del processo di colorazione degli archi, esiste sempre un minimo albero ricoprente che contiene tutti gli archi blu, e non contiene nessun arco rosso. Quindi, alla fine del processo di colorazione, se abbiamo colorato esattamente n-1 archi di blu, avremo ottenuto un MST di G.
- A seconda della scelta della regola da applicare e del taglio/ciclo usato ad ogni passo, si ottengono dal metodo goloso diversi algoritmi con diversi tempi di esecuzione

Teorema dei tagli (regola blu)

Teorema: Dato il grafo G=(V,E,w) non orientato e pesato, e dato un taglio C=(X,Y) in G, un arco e=(u,v) di peso minimo che attraversa il taglio C appartiene sempre ad un qualche MST di G.

Dim. (per assurdo): Supponiamo per assurdo che e non appartenga ad alcun MST di G. Sia T=(V,E′) un qualsiasi MST di G, e consideriamo il taglio C in T.



$w(u,v) \leq w(x,y)$

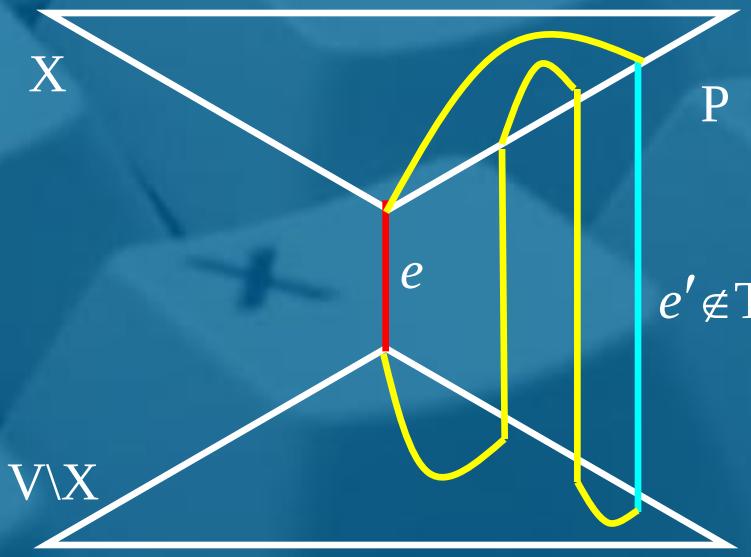
Aggiungendo l'arco e=(u,v) a T ottengo un ciclo in T, e tale ciclo contiene almeno un arco di T che attraversa il taglio.

Allora, l'albero T' ottenuto da T sostituendo uno qualsiasi di tali archi con l'arco (u,v), è un albero ricoprente di G non più pesante di T, che per ipotesi era un MST $\Rightarrow$ T' è un MST di G e (u,v) gli appartiene $\Rightarrow$ contraddizione!

Teorema dei cicli (regola rossa)

Teorema: Sia G=(V,E,w) un grafo non orientato e pesato, sia e l'arco più pesante di un ciclo C in G. Allora esiste almeno un MST di G che non contiene e.

Dim. (per assurdo): Sia e l'arco più pesante in un ciclo $C=\{e\}\cup P$, e supponiamo $e\in T$, un MST di G. Allora, sovrapponendo P a T esisterà almeno un arco e' di P che non appartiene a T e che attraversa il taglio indotto dalla rimozione di e da T (perché altrimenti T non sarebbe aciclico):



Sia $T'=T \setminus \{e\} \cup \{e'\}$. Ovviamente, T′ è un albero ricoprente G. Inoltre, $w(e') \le w(e) \Rightarrow w(T') \le w(T)$

$\Rightarrow$T' è un MST di G che non contiene e!

Algoritmo di Kruskal (1956)

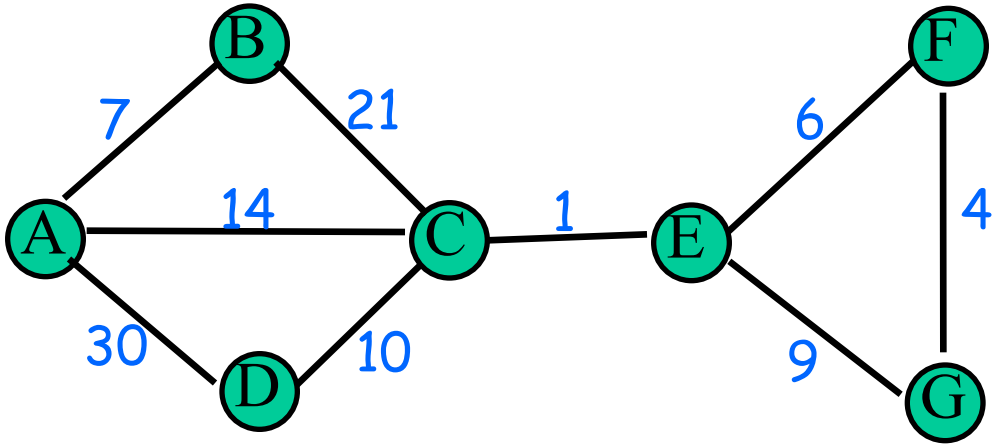
Strategia

- Mantiene una foresta di alberi disgiunti (blu), che all'inizio consiste degli n vertici del grafo
- Per ogni arco, in ordine non decrescente di costo, applica il seguente passo: se l'arco ha entrambi gli estremi nello stesso albero, applica la regola del ciclo e coloralo rosso, altrimenti applica la regola del taglio e coloralo blu
- I vertici nello stesso albero sono mantenuti tramite una struttura dati union/find

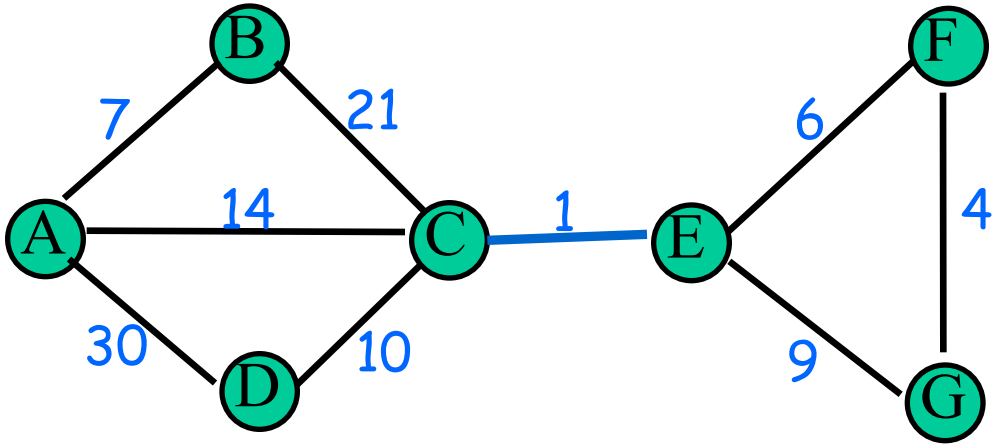
Pseudocodice

```
    algoritmo Kruskal (grafo G) → albero
1.      UnionFind UF
2.      T ← albero vuoto
3.      ordina gli archi di G = (V, E, w) secondo costi non decrescenti
4.      for each ( vertice v in G ) do UF.makeSet(v)
5.      for each ( arco (x, y) in G in ordine non decrescente di costo ) do
6.          T_x ← UF.find(x)
7.          T_y ← UF.find(y)
8.          if ( T_x ≠ T_y ) then
9.              UF.union(T_x, T_y)
10.             aggiungi l'arco (x, y) a T
11.     return T
```

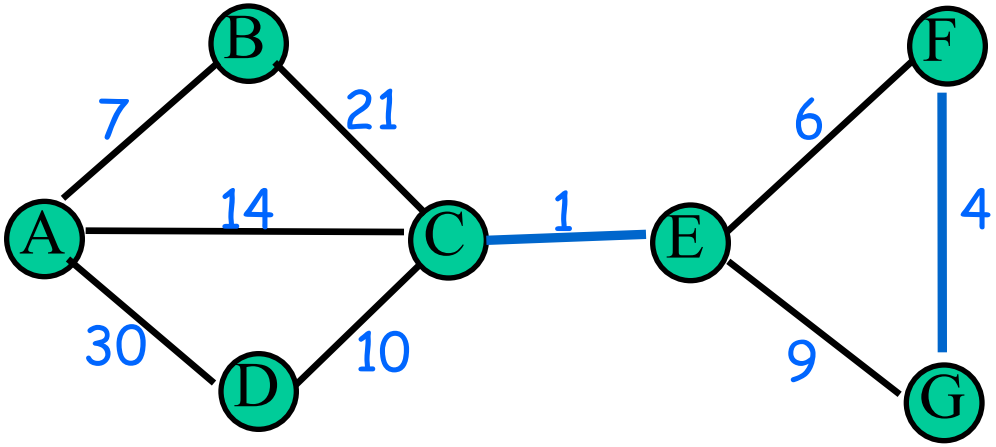
Esempio



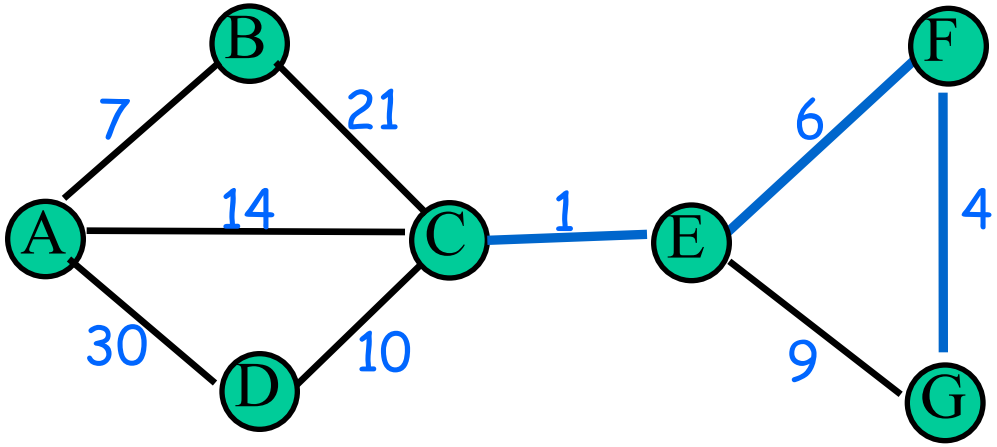
Esempio



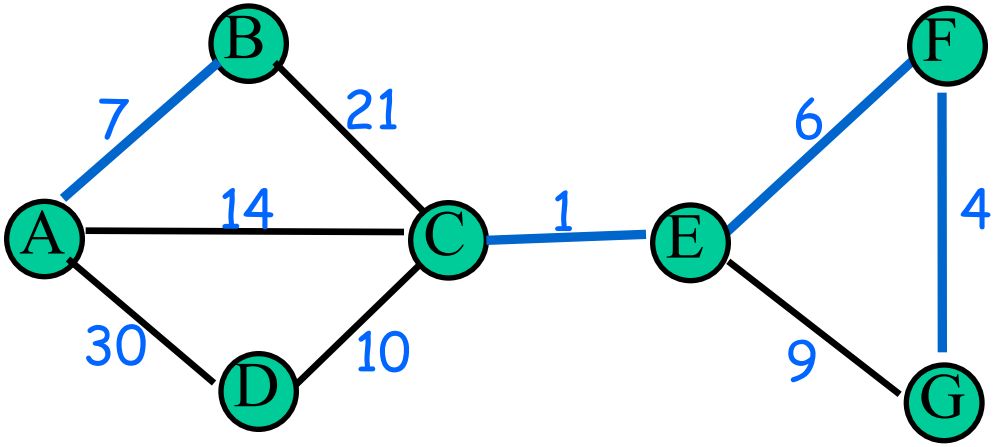
Esempio



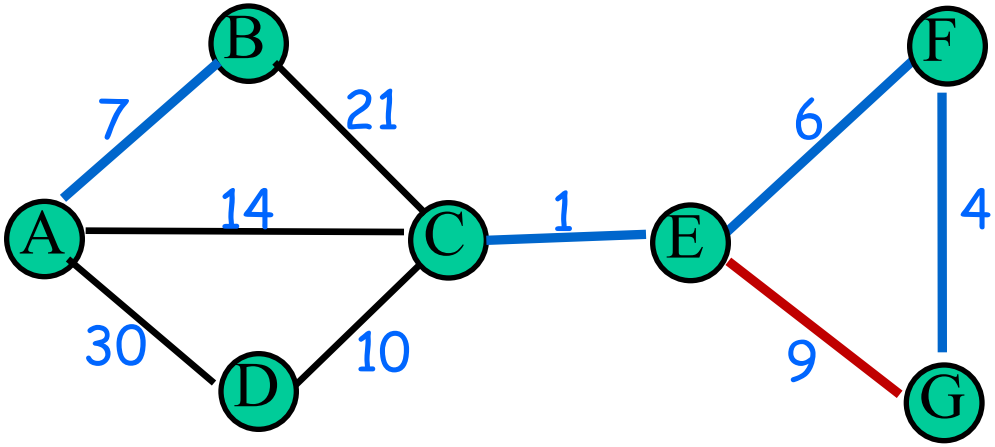
Esempio



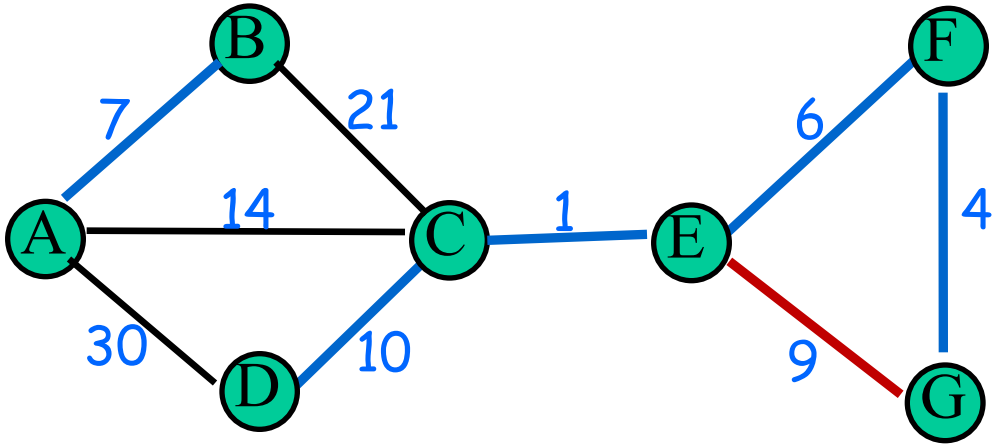
Esempio



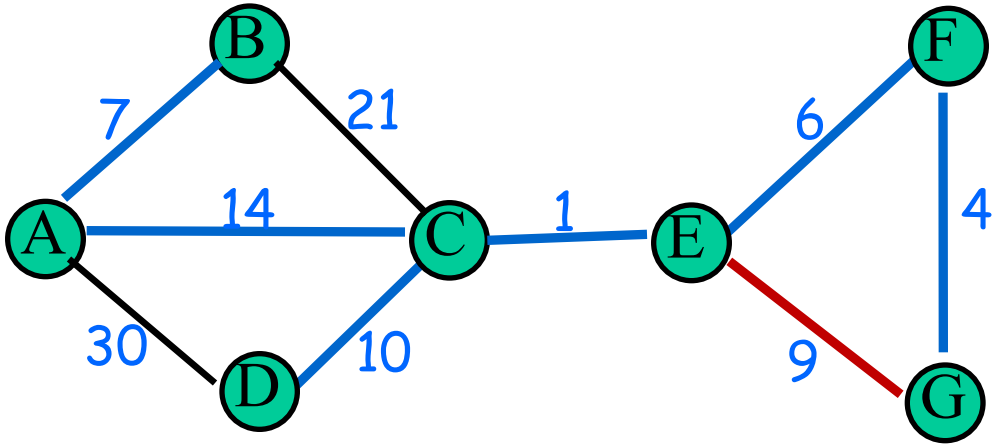
Esempio



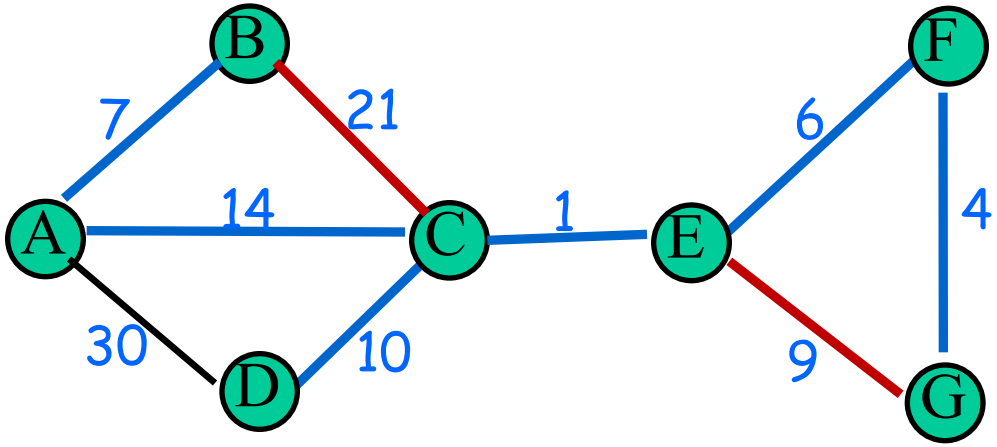
Esempio



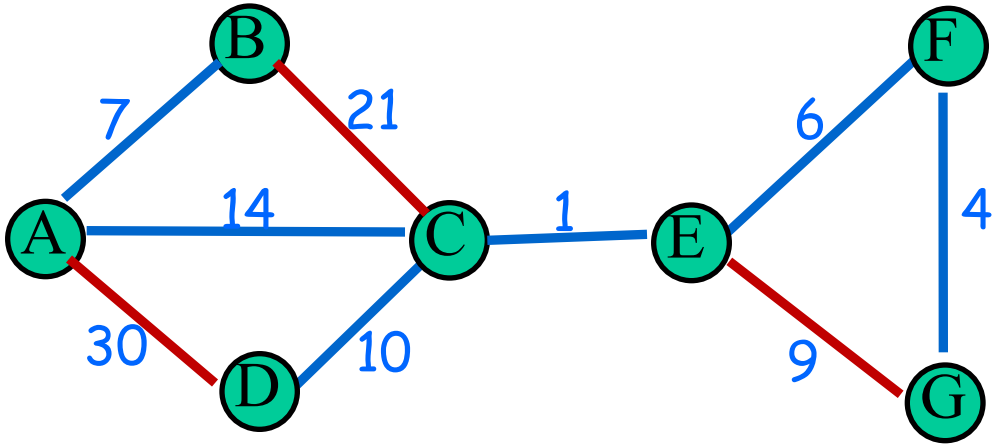
Esempio



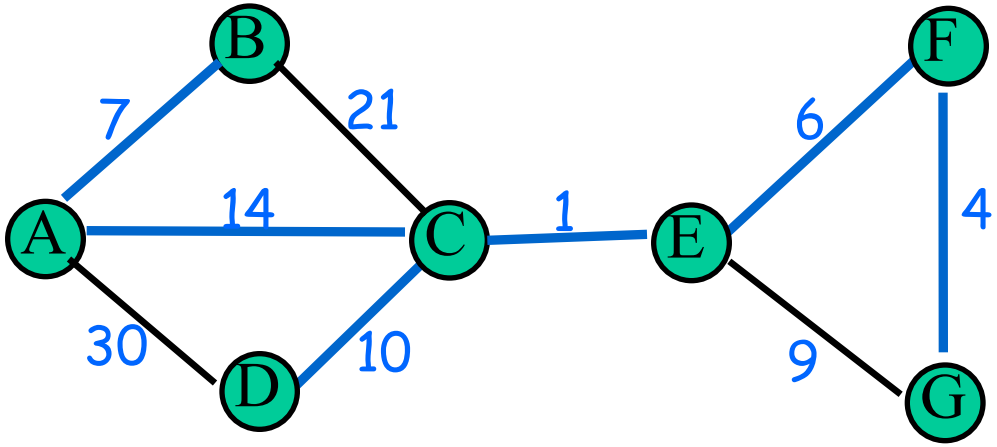
Esempio



Esempio

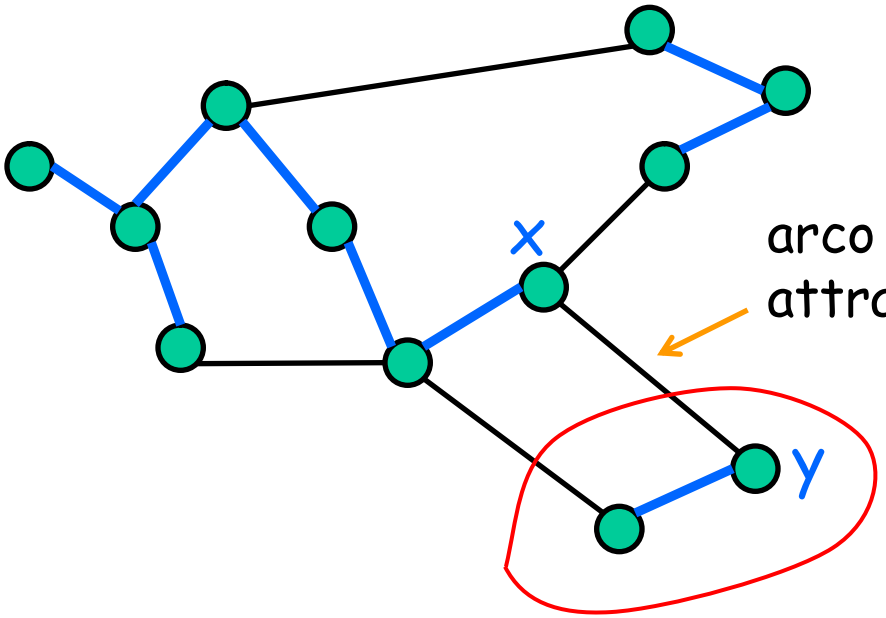


Esempio



Correttezza

quando coloro di blue un arco (x,y)

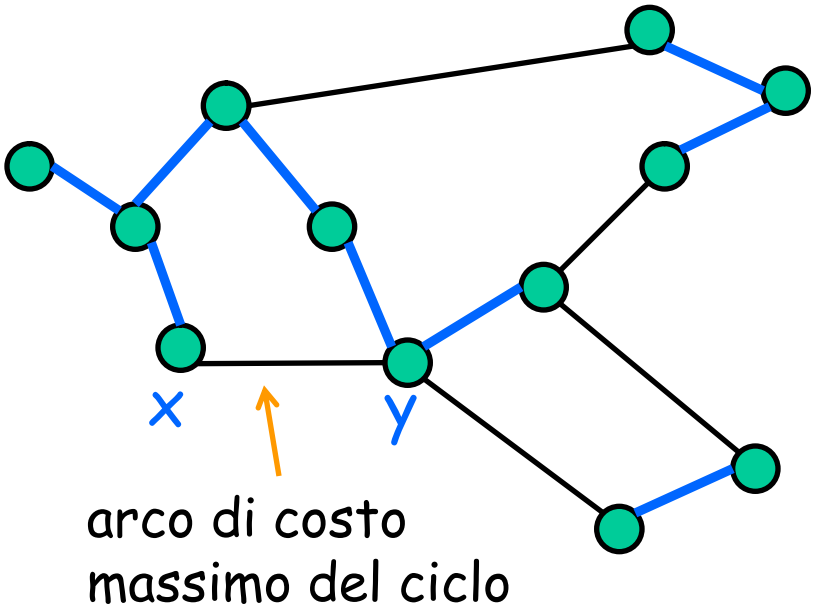


l'algoritmo guarda gli archi
in ordine crescente di peso

arco di costo minimo che
attraversa il taglio

Correttezza

quando coloro di rosso un arco (x,y)



l'algoritmo guarda gli archi
in ordine crescente di peso

Analisi della complessità

Su un grafo con m archi ed n nodi, si eseguono:

- Un ordinamento su m elementi (costo $O(m \log m) = O(m \log n^2) = O(m \log n)$, nell'ipotesi che il grafo in input sia rappresentato tramite una lista di adiacenza);
- n operazioni di Makeset (costo $\Theta(n)$);
- $2m$ operazioni di Find;
- $n-1$ operazioni di Union.

$$\Rightarrow T(n,m)=O(m \log n + n + T(UF(n,m))=$$

$$O(m \log n + T(UF(n,m)))$$

Analisi della complessità

La complessità dipende da come viene risolto UF(n,m):

1. Alberi QuickFind con euristica dell'unione bilanciata:

$$T(UF(n,m))=O(n \log n + m)$$

$$\Rightarrow T(n,m)=O(m \log n + n \log n + m)=O(m \log n).$$

2. Alberi QuickUnion con euristica dell'unione bilanciata:

$$T(UF(n,m))=O(n + m \log n)=O(m \log n)$$

$$\Rightarrow T(n,m)=O(m \log n + m \log n)=O(m \log n).$$

Analisi della complessità

Il tempo di esecuzione dell'algoritmo di Kruskal è O(m log n) nel caso peggiore

(Utilizzando un algoritmo di ordinamento ottimo e gestendo la struttura dati union-find con alberi QuickFind con euristica di unione bilanciata, o alberi QuickUnion con euristica di unione bilanciata (*by rank* o *by size*))

Esercizi

1) Come cambia la complessità dell'algoritmo di Kruskal se si mantengono gli insiemi disgiunti con la struttura dati di Tarjan che usa le uristiche di *union by rank* e *compressione dei cammini*?
2) Assumi che i pesi degli archi di G siano interi compresi fra 1 e k e $k=O(n^c)$, per una qualche costante c. Dare un algoritmo che calcola un MST in tempo o(m log n)
3) Dimostrare che se i pesi degli archi del grafo sono distinti, allora l'MST di G è unico