Algoritmi e Strutture Dati

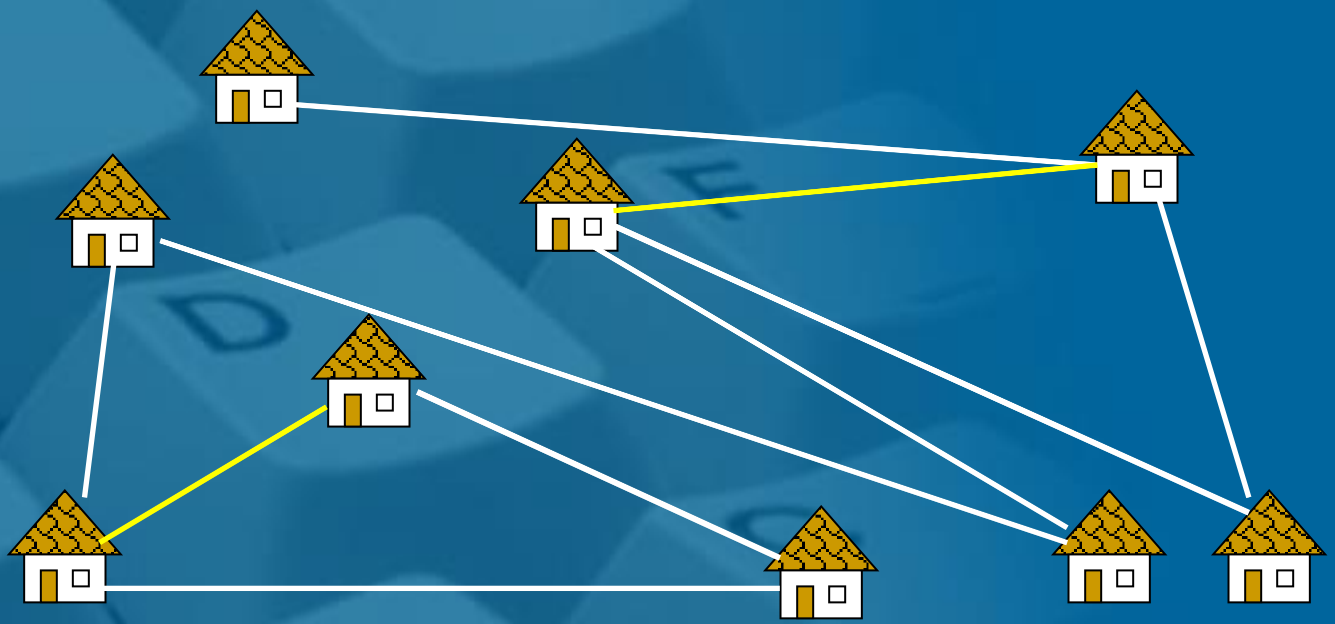
Capitolo 12
Minimo albero ricoprente:
Algoritmo di Kruskal (*)

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.

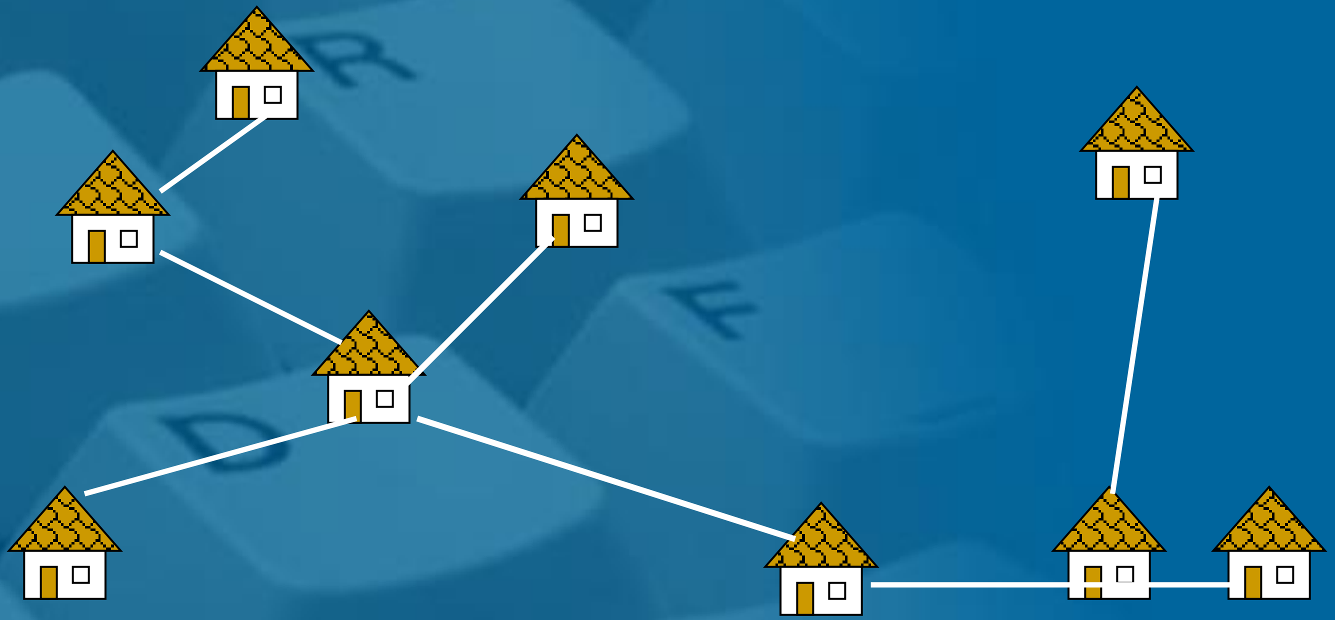
Requisito minimo: connettività tra tutte le abitazioni.

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



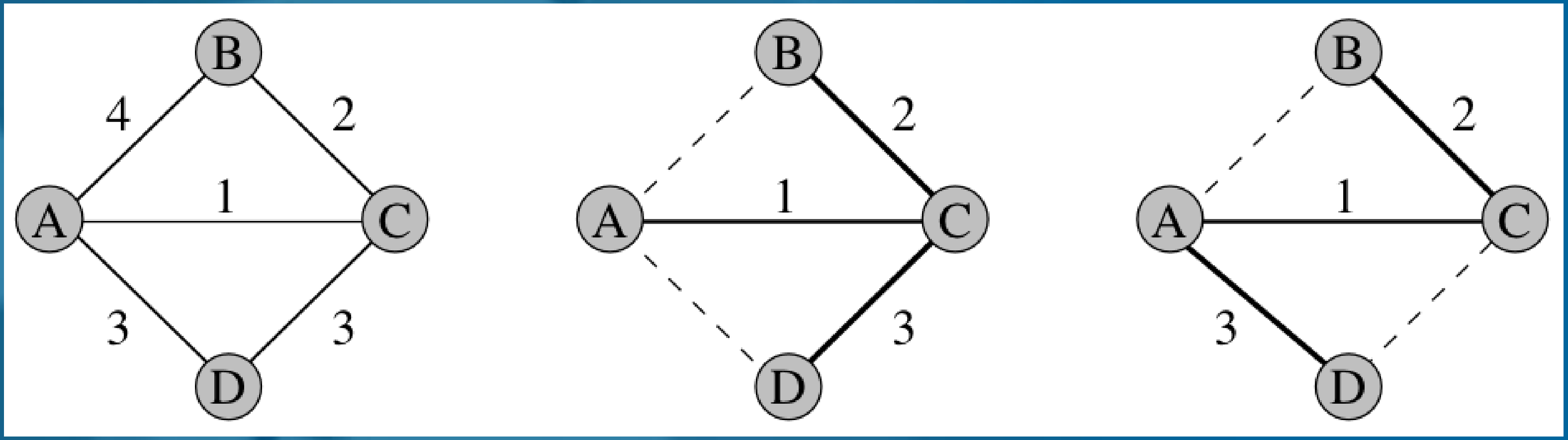
Usa il minimo numero di archi (pari al numero di abitazioni meno 1), e la loro lunghezza complessiva è minima (fidatevi!). In termini teorici, è un minimo albero ricoprente del grafo completo euclideo avente per nodi le abitazioni, e per pesi degli archi la distanza euclidea tra i relativi estremi.

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 di G è un sottografo $T=(V,E'\subseteq E)$ 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 degli archi appartenenti all'albero.
- Un minimo albero ricoprente (MAR) di G è un albero ricoprente di G avente costo minimo.

Esempi

Il minimo albero ricoprente non è necessariamente unico



Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano

Algoritmi che presenteremo

- **Algoritmo di Kruskal (1956): O(m log n)**
- **Algoritmo di Prim (1957): O(m+ n log n)**
- **Algoritmo di Borůvka (1926) : O(m log n)**

Proprietà dei minimi alberi ricoprenti

La tecnica golosa (o greedy)

- Gli algoritmi che studieremo per il calcolo del MAR faranno tutti uso della cosiddetta tecnica golosa (greedy)
- La tecnica golosa si applica principalmente a problemi di ottimizzazione in cui, dato in input un insieme di elementi, bisogna scegliere un sottoinsieme di essi per costruire una soluzione il cui costo minimizzi o massimizzi una certa funzione obiettivo
- Il paradigma dell'algoritmo goloso è il seguente: inizialmente ordina gli elementi in input in base ad un criterio di appetibilità (da cui il termine goloso), e poi ripete le seguenti operazioni:
  - Ad ogni fase i, la soluzione viene accresciuta selezionando l' i-esima componente della stessa: tale componente, tra tutte quelle ammissibili, risulta la migliore in questo momento rispetto al criterio di appetibilità;
  - Una volta fatta la scelta per la i-esima componente, si aggiornano (eventualmente) le appetibilità degli elementi rimanenti, e si passa a considerare le scelte successive, senza più tornare sulla decisione presa.
- Come per la programmazione dinamica, anche in questo caso la tecnica può funzionare solo se il problema gode della proprietà di sottostruttura ottima (sebbene questa sia una condizione necessaria, ma non sufficiente)
- Esempio di algoritmo greedy: Dikstra

Tagli e cicli

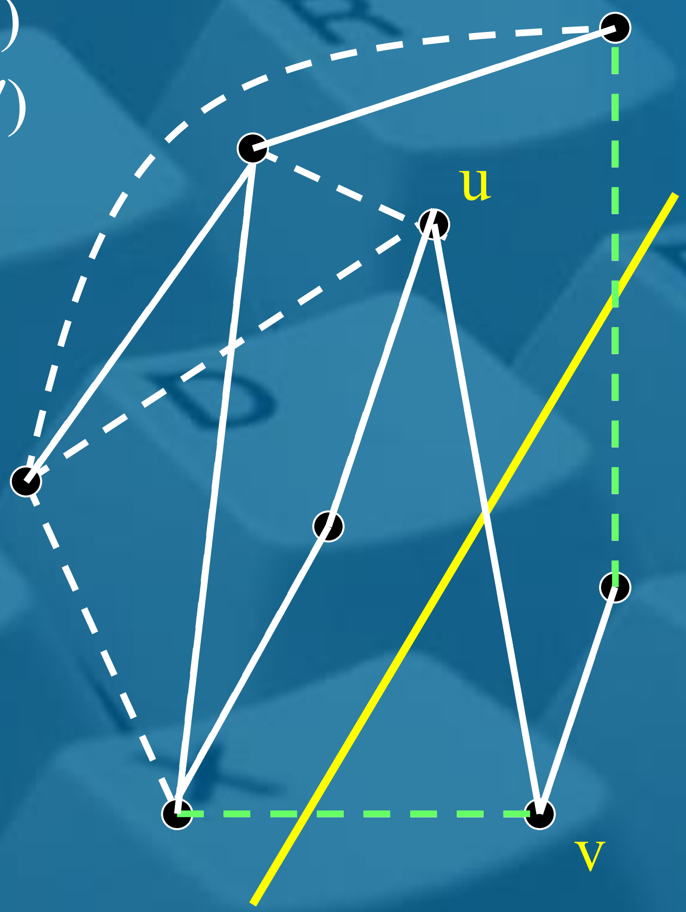
- Nel caso del MAR, i vari algoritmi golosi si baseranno sulla valutazione del peso degli archi nei tagli e nei cicli del grafo:

1. Dato un grafo non orientato G=(V,E), un taglio C=(X,Y) in G è una partizione dei vertici V in due insiemi (disgiunti): X e Y=V\X.
2. Ricordiamo inoltre che, dato un grafo non orientato G=(V,E), un ciclo in G è un cammino semplice chiuso $<v_0,v_1,v_2,\dots,v_k,v_0>$ in G, in cui cioè non ci sono ripetizioni di vertici a parte il primo e l'ultimo

Tagli e alberi ricoprenti

G=(V,E)

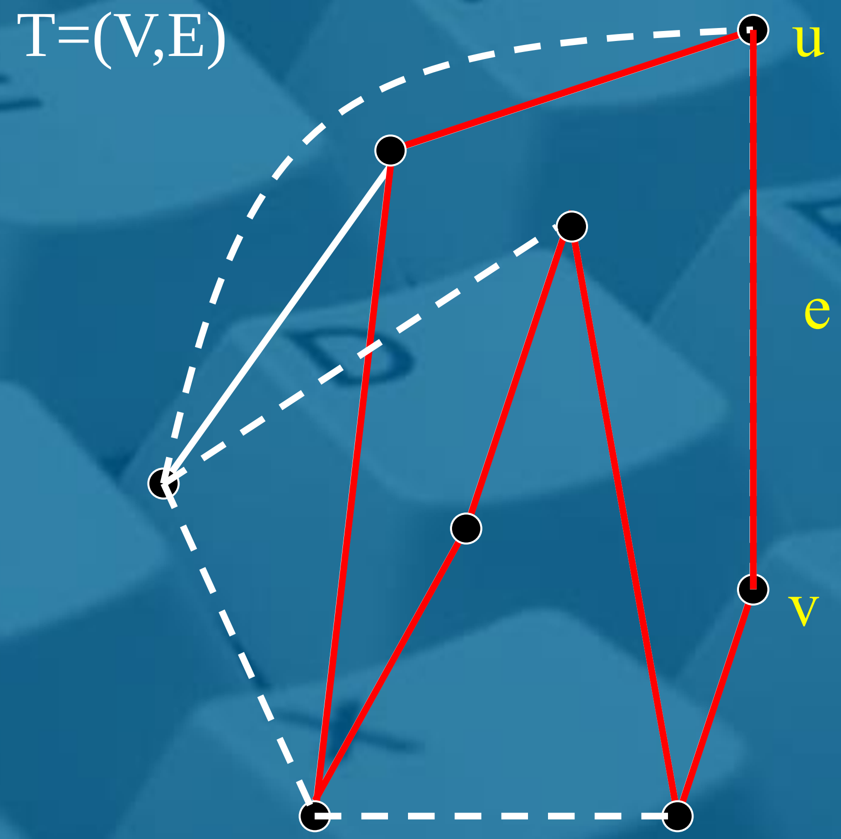
T=(V,E′)



La rimozione di un arco (u,v) da un albero ricoprente T di G **induce un taglio** in G, ovvero quello indotto dai due sottoalberi in cui si partiziona T; gli archi colorati in verde vengono detti archi di attraversamento del taglio, in quanto hanno i due estremi ciascuno in un insieme della partizione definita dal taglio

Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano

Cicli e alberi ricoprenti



Aggiungendo un arco e=(u,v) ad un albero ricoprente T di G individuo un ciclo in G (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

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, e tra tutti gli archi 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, e tra tutti gli archi che appartengono al ciclo, scegline uno di costo massimo e **coloralo di 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.

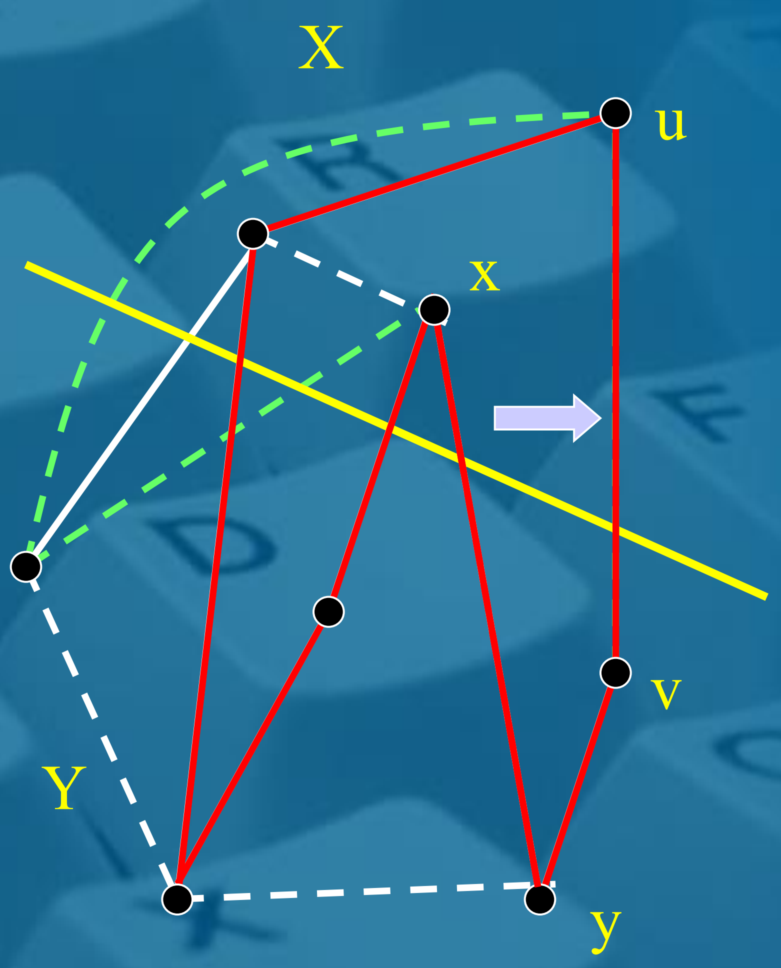
L’approccio “goloso”

- L’approccio goloso applica una delle due regole colorando un arco ad ogni passo, finché tutti gli archi sono colorati; quando un arco assume un colore, lo mantiene per sempre
- Dimostreremo che ad ogni passo del processo di colorazione degli archi, esiste sempre un minimo albero ricoprente di G che conterrà tutti gli archi che sono stati finora colorati di blu, e non conterrà nessun arco che invece è stato colorato di rosso. Quindi, alla fine del processo di colorazione, se abbiamo colorato esattamente n-1 archi di blu, avremo ottenuto un MAR 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 MAR di G.

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



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

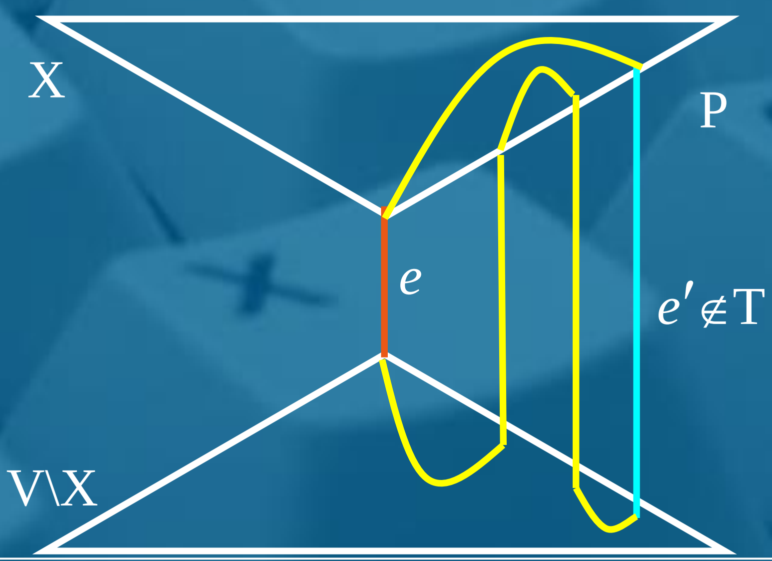
Aggiungendo l'arco e=(u,v) di costo minimo che attraversa C=(X,Y) a T, ottengo un ciclo in T, e tale ciclo contiene almeno un arco di T che attraversa il taglio.

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 MAR ⇒ T′ è un MAR di G e (u,v) gli appartiene ⇒ contraddizione!

Teorema dei cicli (regola rossa)

Teorema: Sia G=(V,E,w) un grafo non orientato e pesato, sia e l'arco strettamente più pesante di un qualsiasi ciclo in G. Allora e non può appartenere ad un MAR di G.

Dim. (per assurdo): Sia e l'arco più pesante in un ciclo $C=\{e\} \cup P$, e supponiamo per assurdo che $e \in T$, un MAR 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,
$w(e') < w(e) \Rightarrow w(T') < w(T)$

$\Rightarrow$ T non è un MAR di G!

Teorema dei cicli (versione estesa)

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

Dim. Esercizio.

Algoritmo di Kruskal (1956)

Strategia

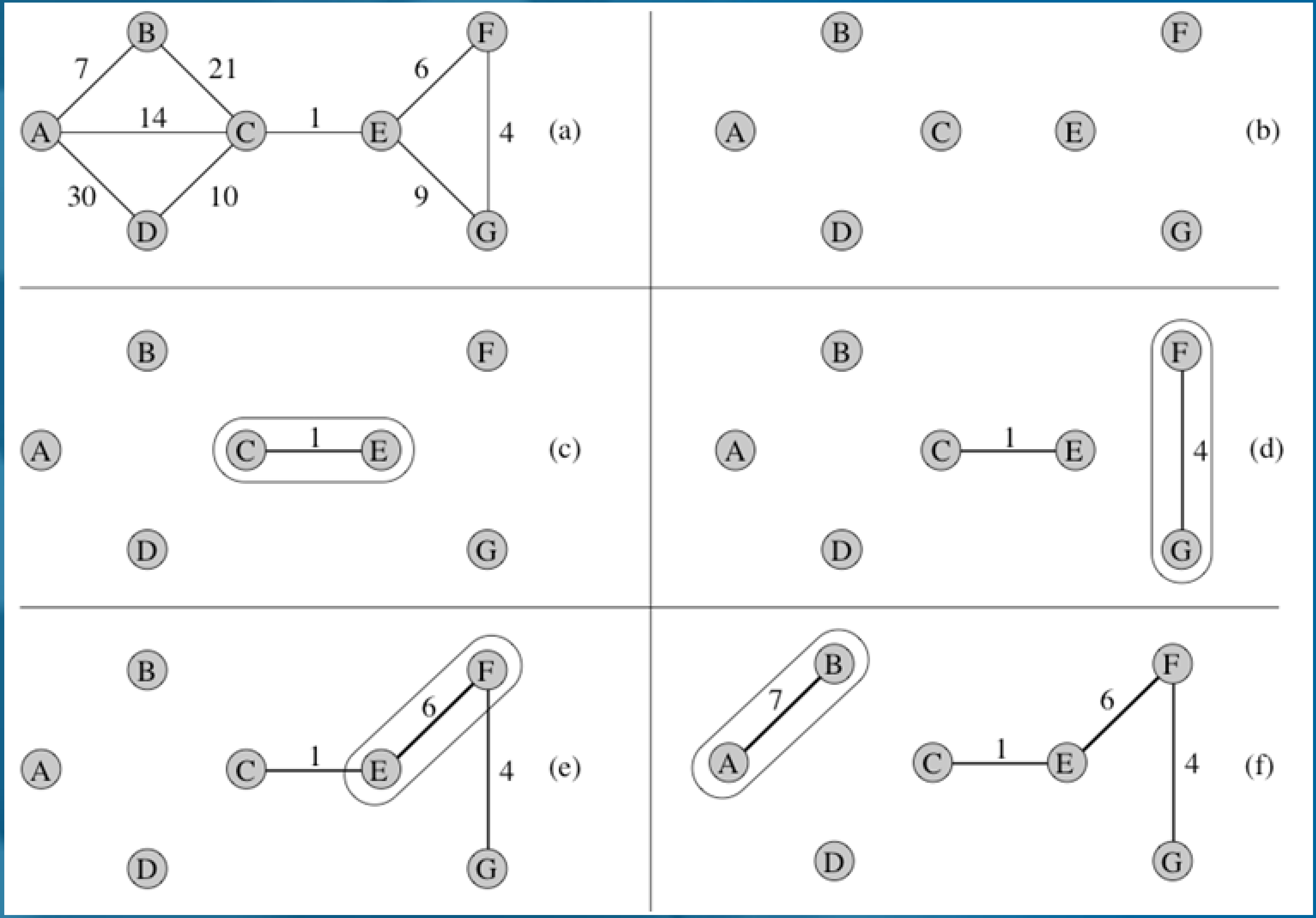
- Mantiene una foresta di alberi disgiunti, che all'inizio consiste degli n vertici del grafo, e che alla fine consisterà di un unico albero, ovvero un MAR del grafo
- Ordina gli archi in ordine non decrescente di costo, e per ogni arco, preso in quest'ordine, applica il seguente passo:
  1. se gli estremi dell'arco appartengono a due alberi diversi della foresta, applica la regola del taglio e aggiorna la soluzione aggiungendo l'arco alla foresta (e quindi unendo i due alberi relativi);
  2. se invece entrambi gli estremi appartengono allo stesso albero, applica la regola del ciclo ed estromettilo dalla soluzione (in sostanza lasciando inalterata la foresta)
- I vertici della foresta sono mantenuti tramite una struttura dati union-find (ove due nodi appartenenti allo stesso albero apparterranno allo stesso insieme)

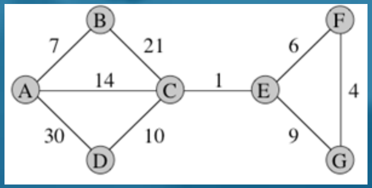
Pseudocodice

algoritmo `Kruskal` $(grafo\ G) \rightarrow albero$
    `UnionFind UF`
    $T = (V, \emptyset)$
    ordina gli archi di $G = (V, E, w)$ secondo costi non decrescenti
    for each (vertice v in G) **do** `UF.makeSet`(v)
    for each (arco (x, y) in G in ordine non decrescente di costo) **do**
        $T_x \leftarrow$ `UF.find`(x)
        $T_y \leftarrow$ `UF.find`(y)
        if ($T_x \neq T_y$) **then**
            `UF.union`(T_x, T_y)
            aggiungi l'arco (x, y) a T
    return T

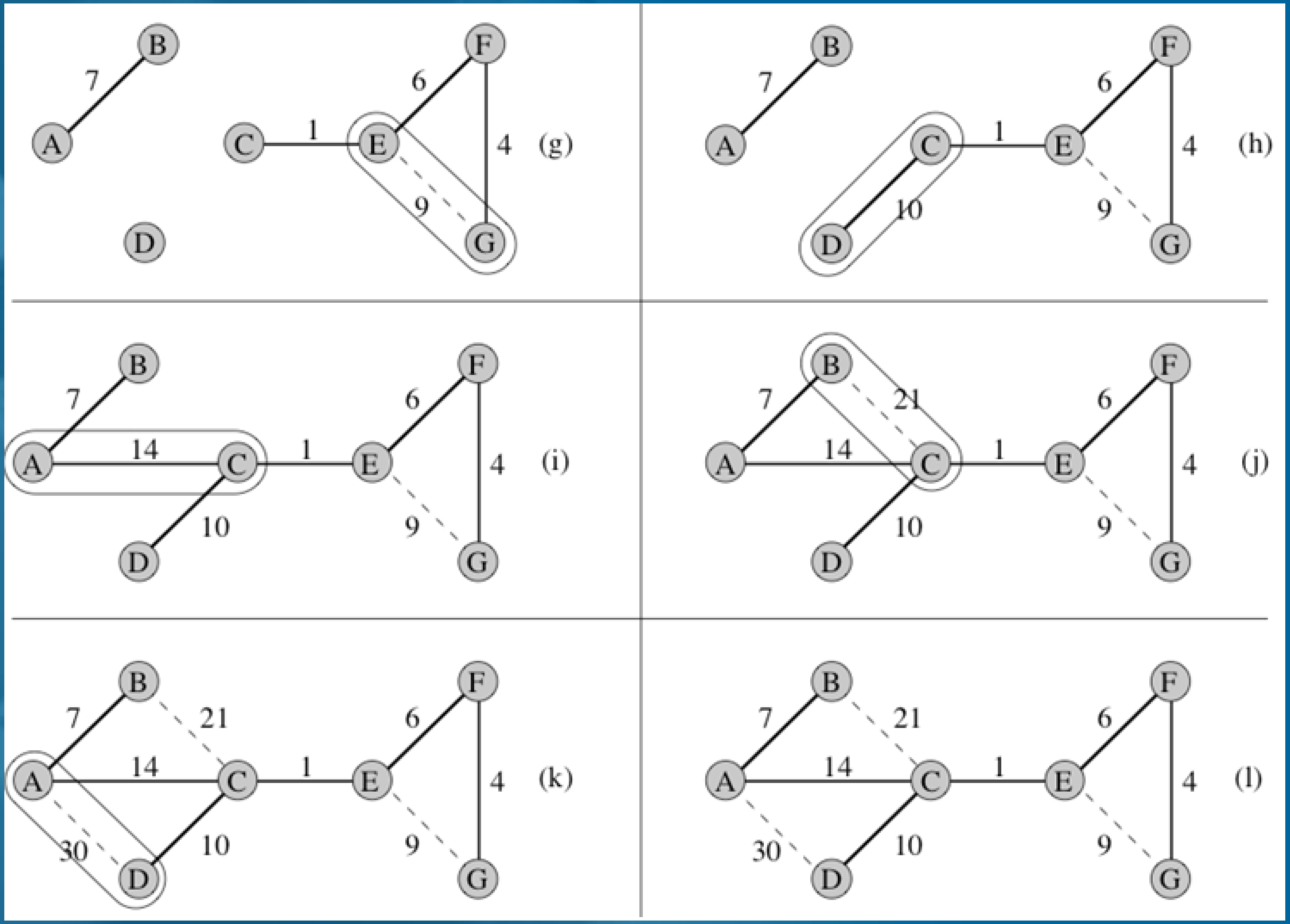
Si noti che durante le operazioni di *Find* abbiamo accesso diretto all'elemento (come avevamo ipotizzato nel problema *Union-Find*) mediante il seguente accorgimento implementativo: durante l'operazione di *Makeset*, ogni elemento viene collegato con un puntatore al corrispondente elemento nella lista di adiacenza nel grafo

Esempio (1/2)





Esempio (2/2)



Analisi della complessità

Dato in input un grafo con m archi ed n nodi, se T(n,m) denota la complessità temporale di Kruskal, si avrà:

- Un ordinamento su m elementi (costo $\Theta(m \log m) = O(m \log n^2) = O(m \log n)$, 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 e n-1 operazioni di union; sia T(UF(n,m)) il costo necessario per eseguire tali operazioni

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

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

Analisi della complessità

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

1. Alberi QuickFind: $T(UF(n,m))=O(n^2 + m)=O(n^2)$

   $\Rightarrow T(n,m) = O(m \log n + T(UF(n,m))) = O(m \log n + n^2)$.

2. Alberi QuickFind con euristica dell'unione bilanciata (*union by size*):

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

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

3. Alberi QuickUnion: $T(UF(n,m))=O(n + m \cdot n)=O(m \cdot n)$

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

4. Alberi QuickUnion con euristica dell'unione bilanciata (*union by rank o by size*):

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

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

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 in un grafo rappresentato mediante liste di adiacenza, e gestendo la struttura dati union-find con alberi QuickFind con euristica di unione bilanciata (*union by size*), o alberi QuickUnion con euristica di unione bilanciata (*union by rank* o *by size*))

Domanda di approfondimento

Confrontare le complessità computazionali delle implementazioni di Kruskal con alberi QuickFind ed alberi QuickUnion (senza euristiche di bilanciamento).