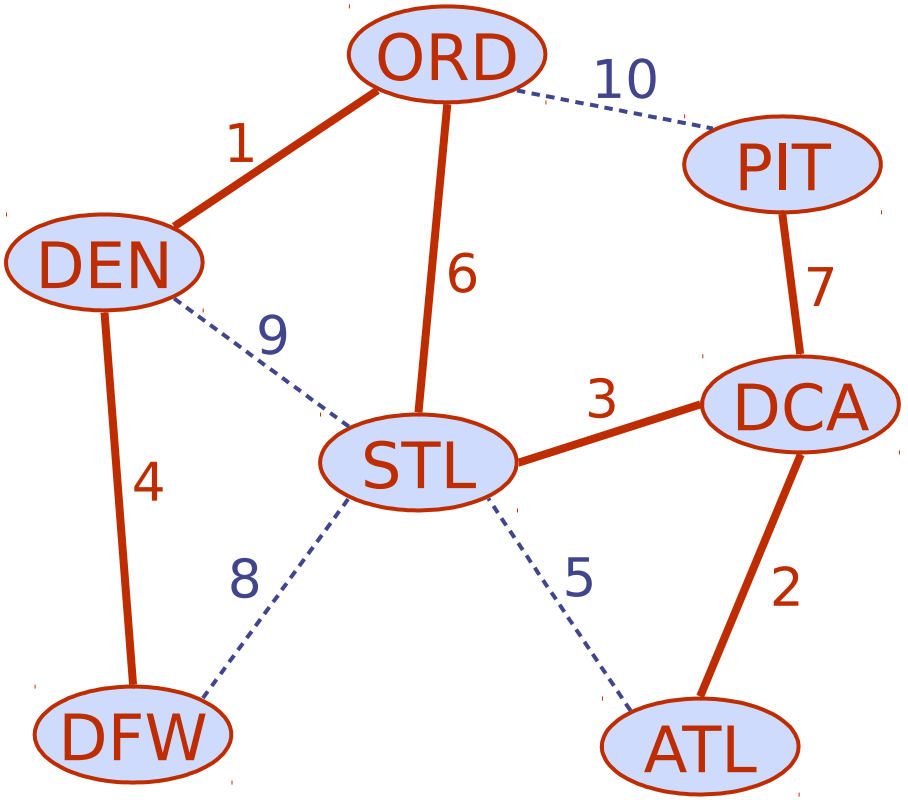
Alberi minimi ricoprenti (Minimum Spanning Tree)

Luca Becchetti

Presentazione tratta dalle slide che accompagnano il testo Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Albero minimo ricoprente

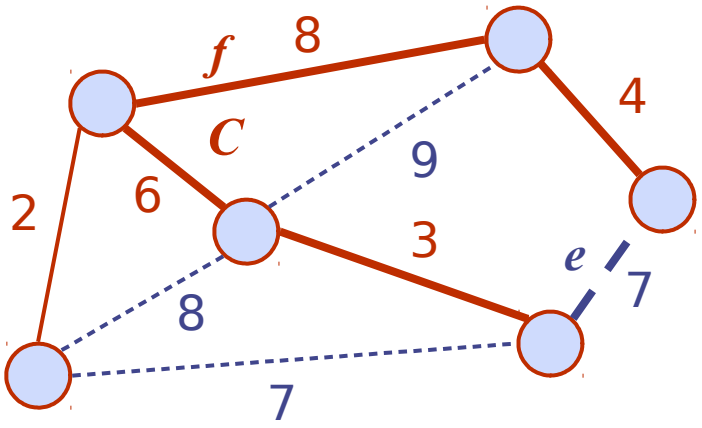
- Consideriamo grafi *non diretti*
  - **Perché?**
- Sottografo ricoprente di G
  - Sottografo che contiene tutti i vertici di G
- Albero ricoprente
  - Qualunque albero che copre tutti i vertici di G
- Albero minimo ricoprente
  - Dato un grafo G (in generale pesato)
  - Albero ricoprente di peso totale minimo
- Applicazioni
  - Reti di comunicazione
  - Reti di trasporto



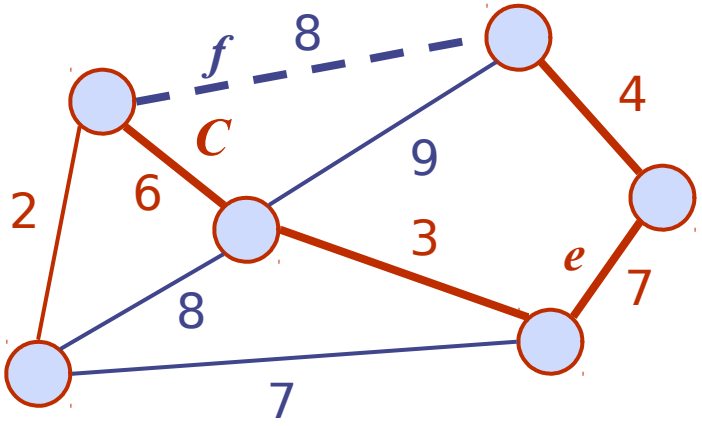
Proprietà

Proprietà di ciclo

- T: un MST di G
- e: arco di G non in T
- C: ciclo presente in T U $\{e\}$
- Per ogni arco f di C
  - w(f) ≤ w(e)
- Prova
  - Per contraddizione
  - Se w(f) > w(e) → nuovo albero di peso minore sostituendo e a f



Sostituendo e a f abbiamo un albero di peso minore

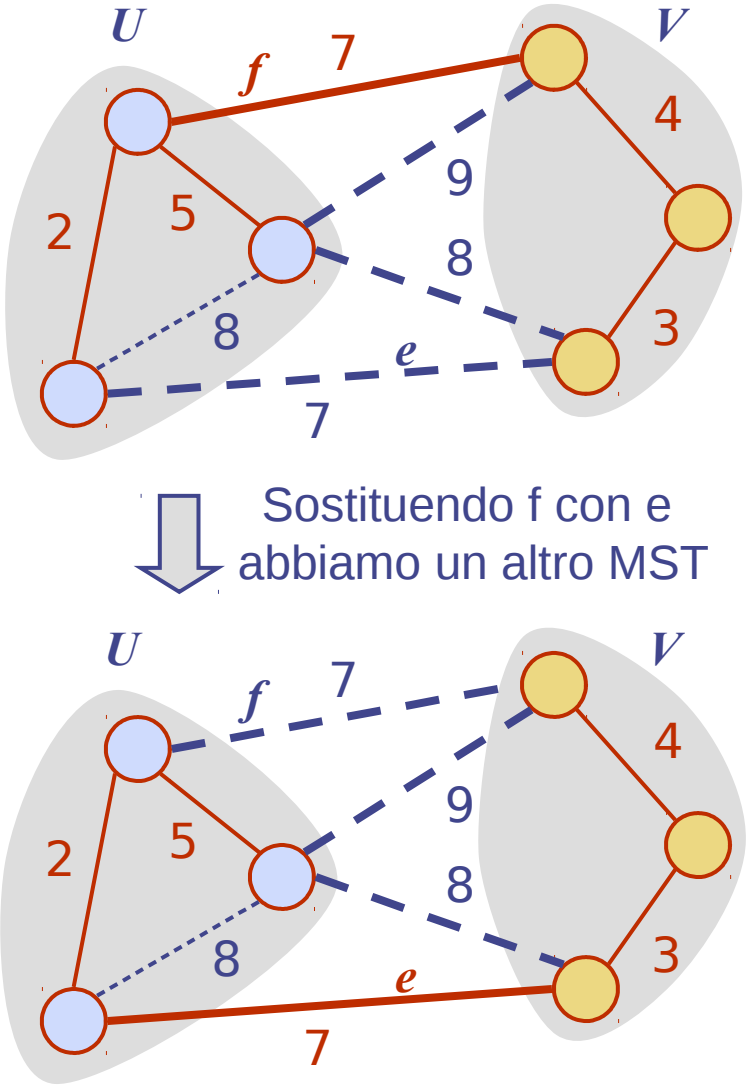


Proprietà di taglio

- Taglio: partizione di V in due sottoinsiemi V_1 e V_2
- Arco del taglio: qualsiasi arco con un estremo in V_1 e l'altro in V_2
- Proprietà di taglio
  - Sia e in arco di peso minimo del taglio (V_1, V_2)
  - Esiste un albero minimo ricoprente che contiene e

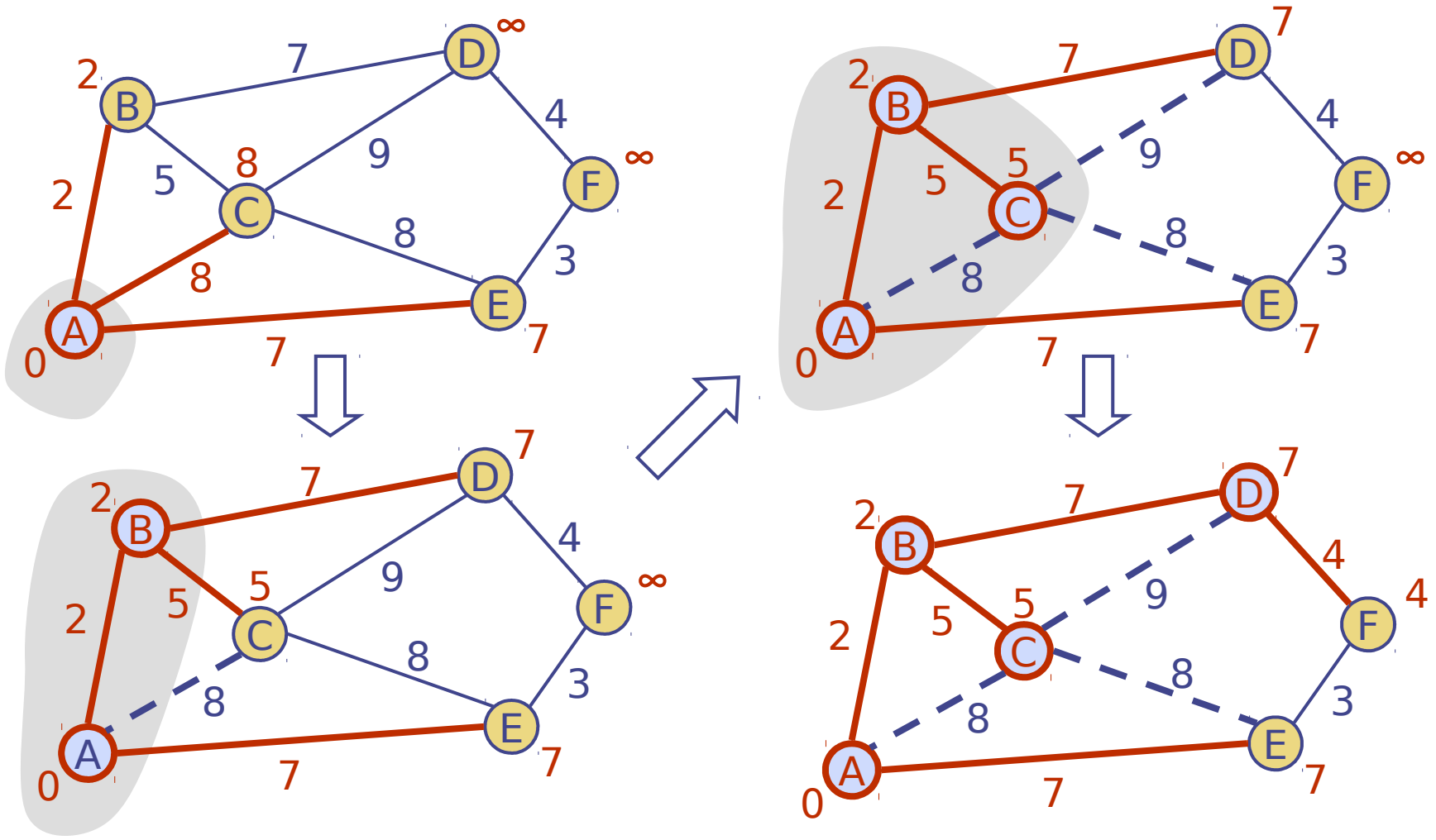
Prova

- Sia T un MST di G
- Se T non contiene e si consideri il ciclo C presente in T U {e}
- Sia f un arco di C appartenente al taglio
- Per la proprietà di ciclo
  - w(f) ≤ w(e)
  - Ma e è minimo → w(f) = w(e) → sostituendo f con e abbiamo ancora un MST

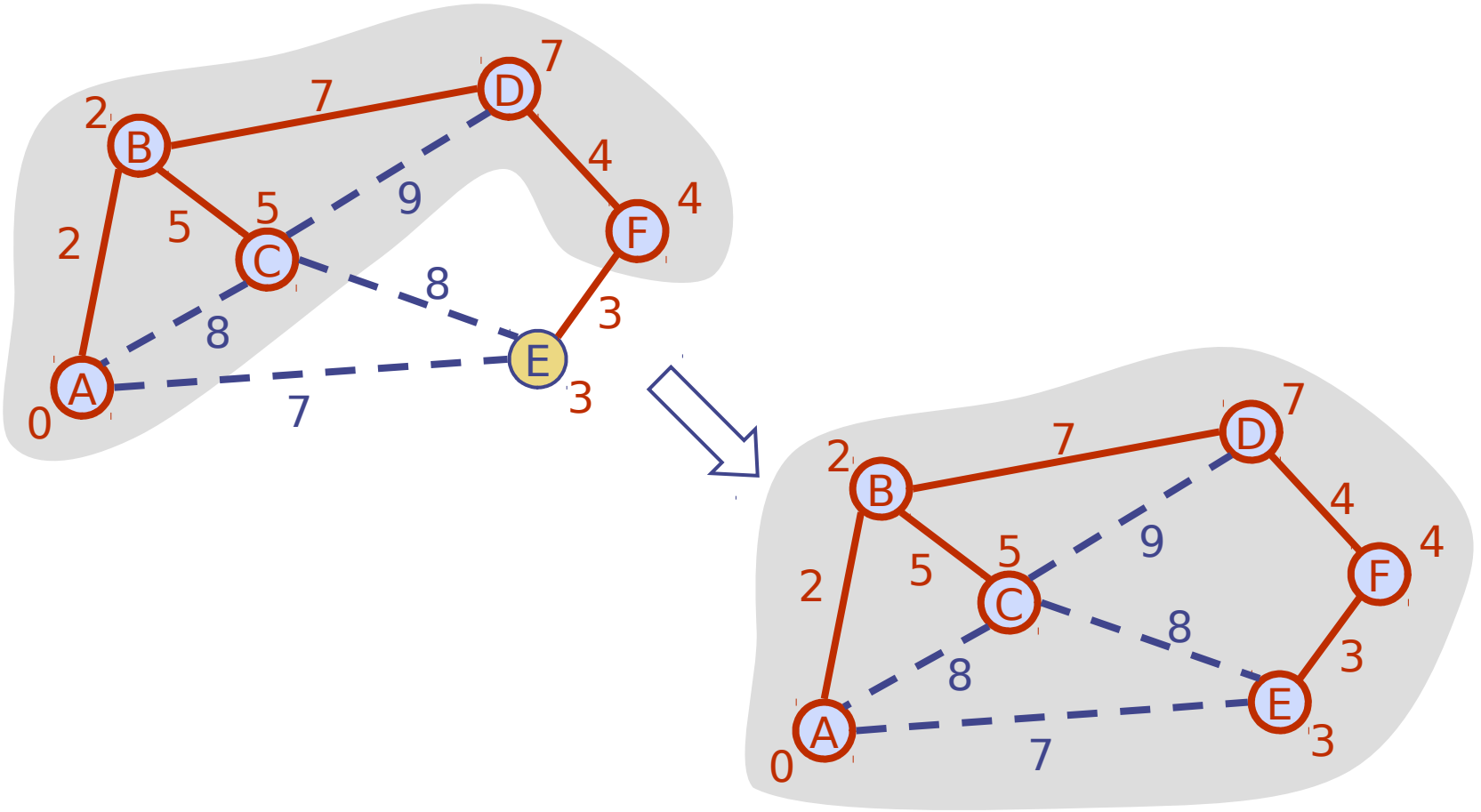


Algoritmo di Prim-Jarnik

Algoritmo di Prim-Jarnik illustrato



Algoritmo di Prim-Jarnik illustrato



Algoritmo di Prim-Jarnik: pseudocodice

Algorithm PrimJarnik(G):
Input: An undirected, weighted, connected graph G with n vertices and m edges
Output: A minimum spanning tree T for G

Pick any vertex s of G
$D[s] = 0$
for each vertex $v \neq s$ **do**
  $D[v] = \infty$
Initialize $T = \emptyset$.
Initialize a priority queue Q with an entry $(D[v], (v, \text{None}))$ for each vertex v, where $D[v]$ is the key in the priority queue, and (v, None) is the associated value.
while Q is not empty **do**
  $(u, e) =$ value returned by Q.remove_min()
  Connect vertex u to T using edge e.
  for each edge $e' = (u, v)$ such that v is in Q **do**
    {check if edge (u, v) better connects v to T}
    if $w(u, v) < D[v]$ **then**
      $D[v] = w(u, v)$
      Change the key of vertex v in Q to $D[v]$.
      Change the value of vertex v in Q to (v, e').
return the tree T

Analisi: complessità

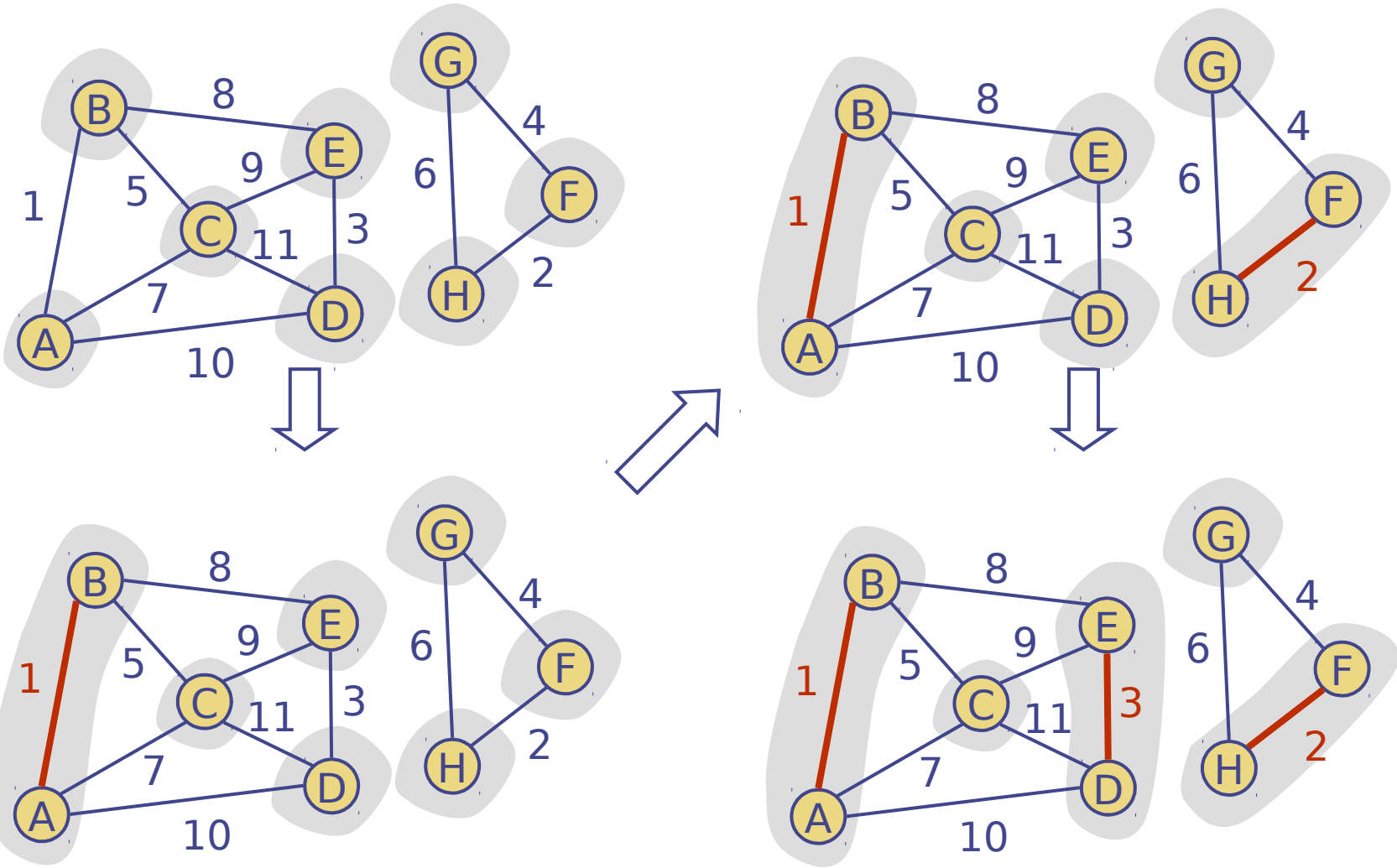
- Costo delle inizializzazioni
  - O(nlog n)
- Costo della generica iterazione del ciclo while (sia u il nodo estratto da Q)
  - O(deg(u)log n) → possibile aggiornamento di Q per ogni vicino di u
- Costo totale
  - O((n + m)log n)

Analisi - correttezza

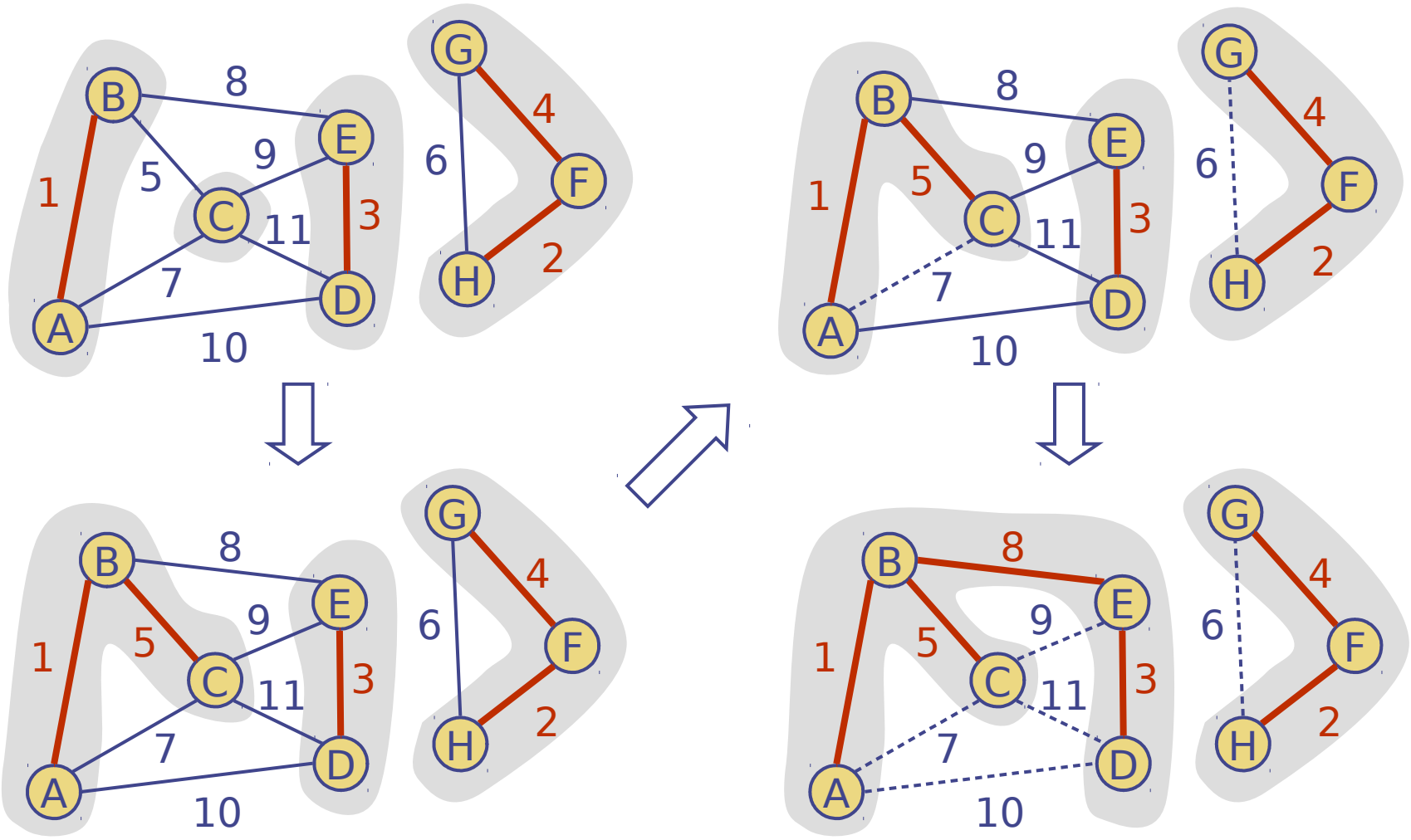
- Supponiamo per semplicità che tutti i pesi degli archi siano diversi tra loro
- Ogni iterazione del ciclo while identifica un taglio
  - I vertici in Q e quelli in V – Q
- Viene sempre selezionato l’arco di peso minimo del taglio
  - Questo arco *deve* appartenere all’MST per la proprietà di taglio

Algoritmo di Kruskal

Algoritmo di Kruskal illustrato



Algoritmo di Kruskal illustrato



Algoritmo di Kruskal: pseudocodice

Algorithm Kruskal(G):

Input: A simple connected weighted graph G with n vertices and m edges

Output: A minimum spanning tree T for G

for each vertex v in G **do**

Define an elementary cluster $C(v) = \{v\}$.

Initialize a priority queue Q to contain all edges in G, using the weights as keys.

$T = \emptyset$ {T will ultimately contain the edges of the MST}

while T has fewer than $n-1$ edges **do**

(u,v) = value returned by Q.remove_min()

Let $C(u)$ be the cluster containing u, and let $C(v)$ be the cluster containing v.

if $C(u) \neq C(v)$ **then**

Add edge (u,v) to T.

Merge $C(u)$ and $C(v)$ into one cluster.

return tree T

Union find

Analisi

- Complessità
  - Inizializzazioni: O(mlog n)
  - Ciclo while: abbiamo una sequenza di al più n - 1 operazioni Union-Find → costo O(n + nlog n)
  - Costo complessivo
    - O((m + n)log n)
- Correttezza
  - Stesso argomento usato per algoritmo di Prim-Jarnik