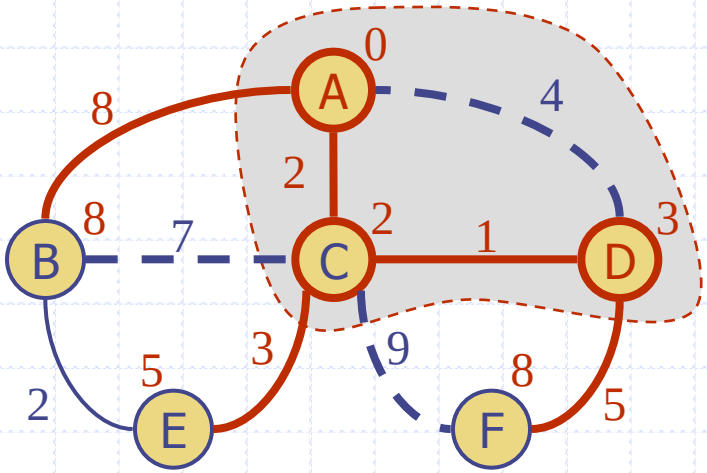
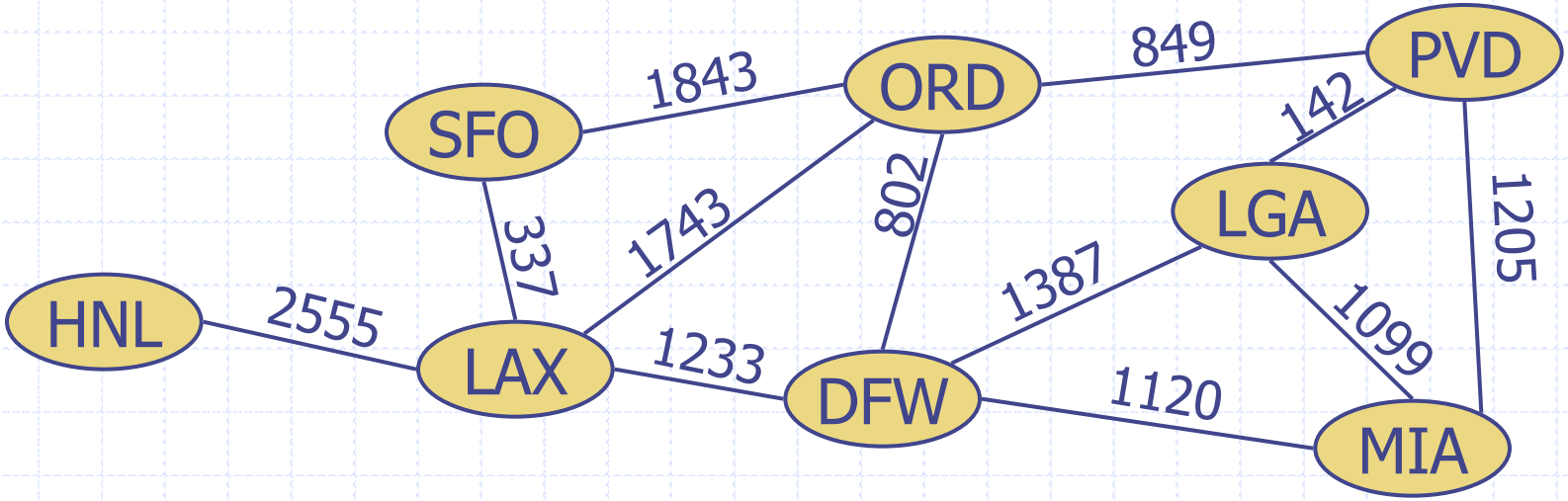
Cammini Minimi



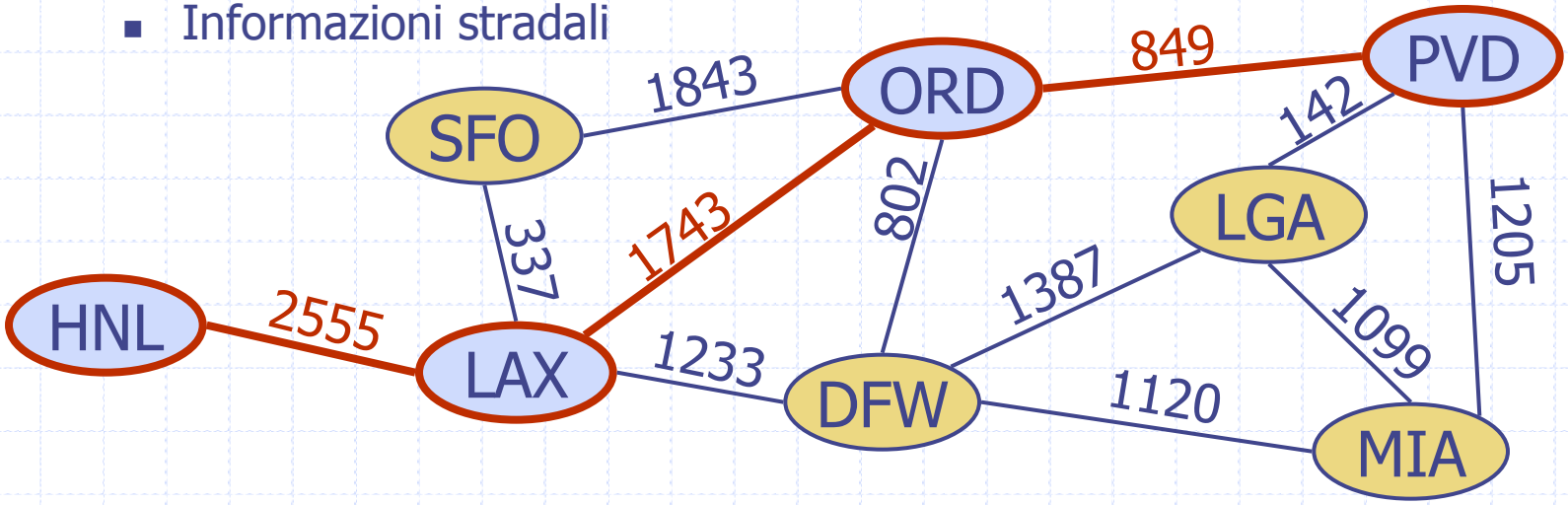
Grafi Pesati

- In un grafo non pesato, ogni arco ha associato un valore numerico, chiamato peso dell'arco
- I pesi dell'arco possono rappresentare distanze, costi, etc.
- Esempio:
  - In un grafo di rotte aeree, il peso di un arco rappresenta la distanza in miglia tra gli aereoporti terminali



Cammini Minimi

- Dato un grafo pesato e due vertici u e v, vogliamo trovare un cammino di minimo costo totale tra u e v.
  - Lunghezza di un cammino e' la somma dei pesi degli archi
- Esempio:
  - Cammini minimi tra Providence and Honolulu
- Applicazioni
  - Routing di pacchetti in Internet
  - Penotazione di aerei
  - Informazioni stradali



Proprieta' dei Cammini Minimi

Proprieta' 1:

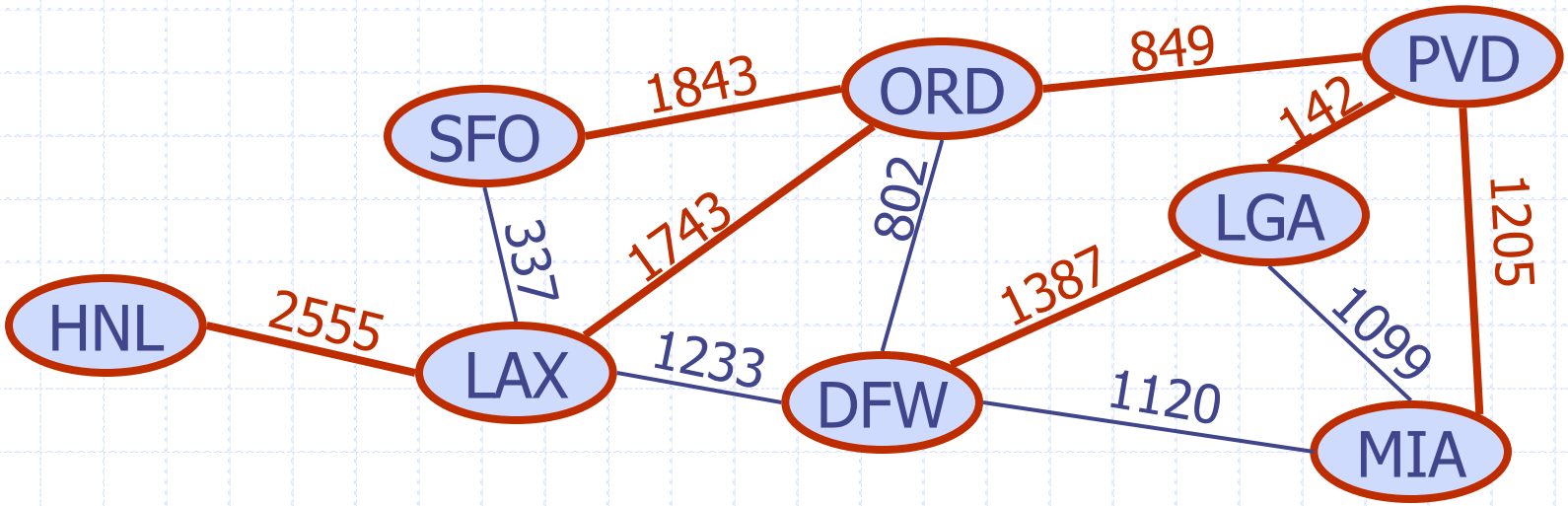
Un sottocammino di un cammino minimo e' un cammino minimo

Proprieta' 2:

L'insieme dei cammini minimi da un vertice di partenza a tutti gli altri vertici forma un albero

Esempio:

Albero dei cammii minimi da Providence



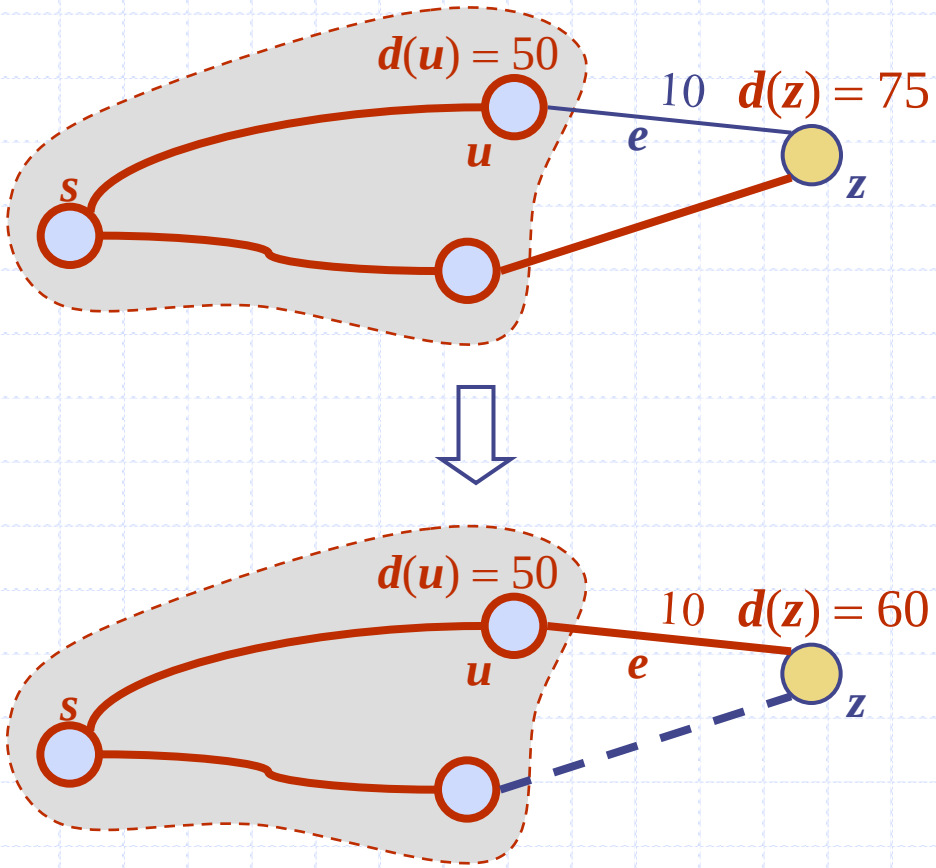
Algoritmo di Dijkstra

- La distanza di un vertice v da un vertice s e' la lunghezza del cammino minimo tra s e v
- L'algoritmo di Dijkstra calcola le distanze di tutti i vertici da un dato vertice di partenza s
- Assunzioni:
  - il grafo e' connesso
  - gli archi sono non diretti
  - i pesi degli archi sono non negativi
- Cresciamo una "**nuvola**" di vertici che si espande da s fino a coprire tutti i vertici
- Memorizziamo per ogni vertice v un'etichetta $d(v)$ rappresentante la distanza da s a v nel sottografo formato dalla nuvola e dai suoi vertici adiacenti
- Ad ogni passo
  - Aggiungiamo alla nuvola il vertice u al di fuori della nuvola con la minima etichetta di distanza $d(u)$
  - Aggiorniamo le etichette dei vertici adiacenti a u

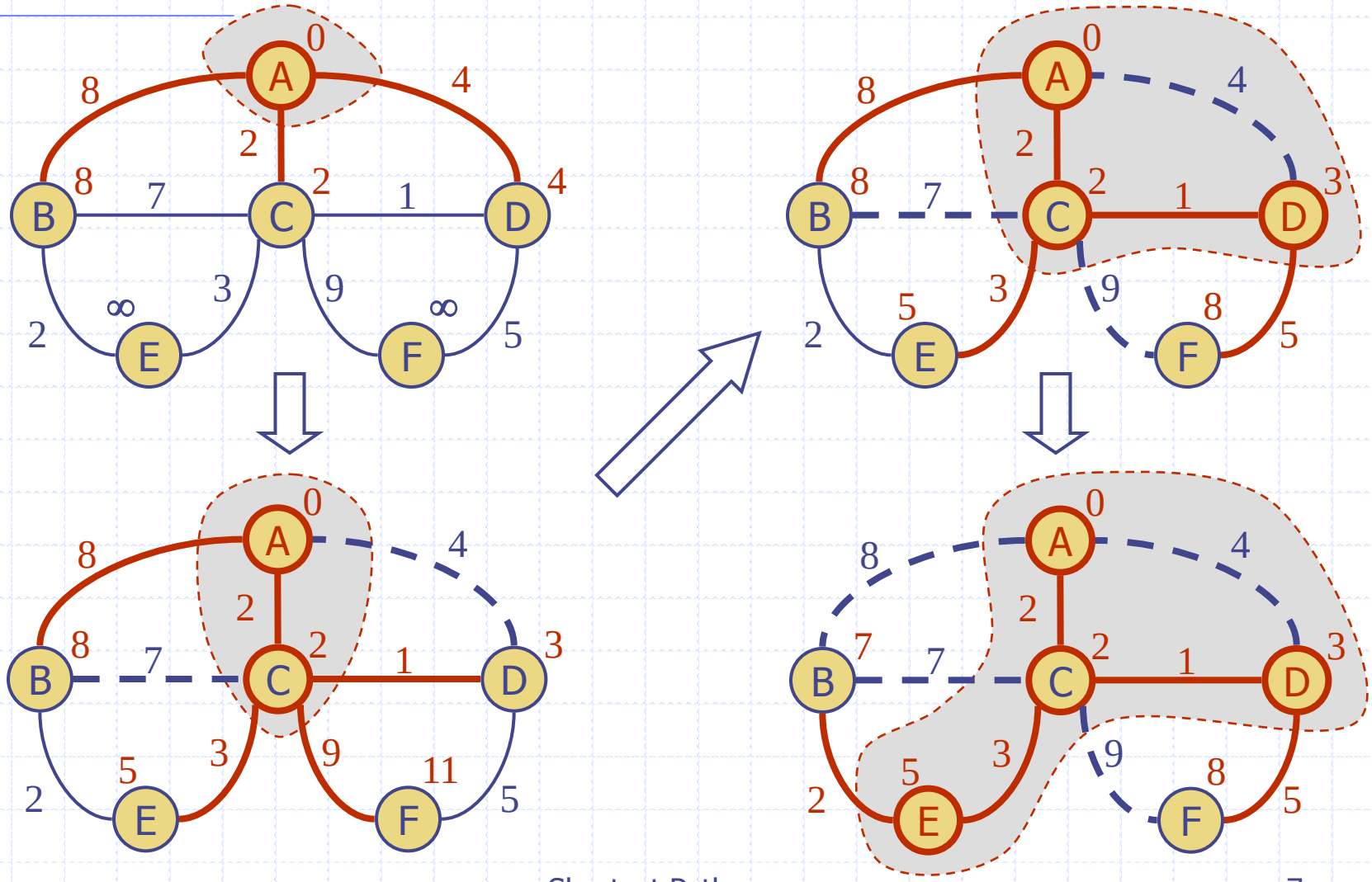
Rilassamento di un Arco

- Considera un arco $e = (u,z)$ tale che
  - u e' il vertice aggiunto piu' recentemente alla nuvola
  - z non e' nella nuvola
- Il rilassamento di un arco e aggiorna la distanza $d(z)$ come segue :

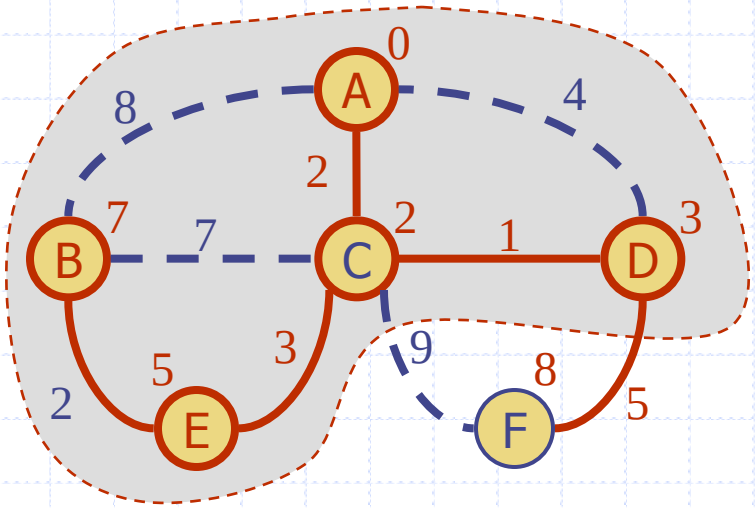
$$d(z) \leftarrow \min\{d(z), d(u) + weight(e)\}$$

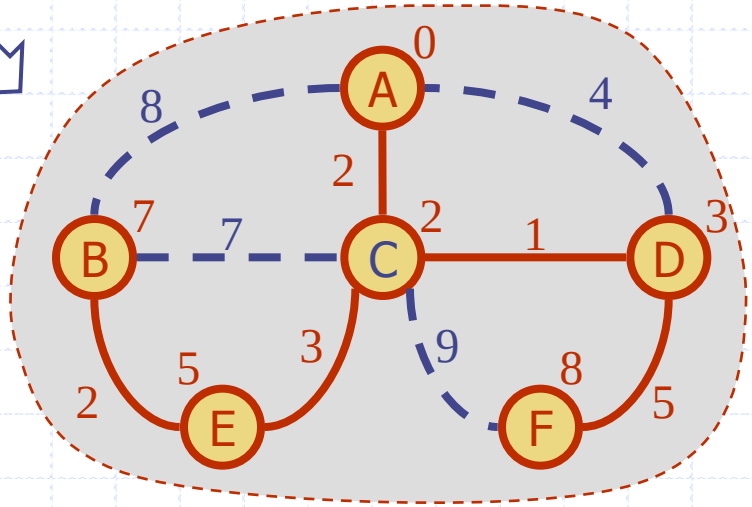


Esempio



Esempio (cont.)





Algoritmo di Dijkstra

- Una coda di priorita' memorizza i vertici fuori della nuvola
  - Chiave: distanza
  - Elemento: vertice
- Metodi basati su Locator
  - *insert*(*k*,*e*) restituisce un locator
  - *replaceKey*(*l*,*k*) modifica la chiave di un elemento
- Memorizziamo due etichette per ogni vertice v:
  - Distanza (d(v) etichetta)
  - locator nella coda di priorita'

```
Algorithm DijkstraDistances(G, s)
    Q ← new heap-based priority queue
    for all v ∈ G.vertices()
        if v = s
            setDistance(v, 0)
        else
            setDistance(v, ∞)
        l ← Q.insert(getDistance(v), v)
        setLocator(v,l)
    while ¬Q.isEmpty()
        u ← Q.removeMin()
        for all e ∈ G.incidentEdges(u)
            { relax edge e }
            z ← G.opposite(u,e)
            r ← getDistance(u) + weight(e)
            if r < getDistance(z)
                setDistance(z,r)
                Q.replaceKey(getLocator(z),r)
```

Analisi dell'Algoritmo di Dijkstra

- Operazioni su Grafi
  - Metodo incidentEdges chiamato una volta per ogni vertice
- Operazioni sulle etichette
  - Assegna/accede $O(\deg(z))$ volte le etichette distanza e locator di un vertice z
  - Assegna/accede un'etichetta in tempo $O(1)$
- Operazioni sulla coda di priorita'
  - Ogni vertice e' inserito e rimosso una volta dalla coda di priorita', dove ogni inserimento o rimozione ha costo $O(\log n)$
  - La chiave di un vertice nella coda di priorita' e' modificata al massimo $\deg(w)$ volte, dove ogni modfica di chiave costa $O(\log n)$
- Dijkstra ha costo $O((n + m) \log n)$ se il grafo e' rappresentatp con una lista di adiacenza
  - Ricorda che $\sum_v \deg(v) = 2m$
- Il tempo di esecuzione e' anche $O(m \log n)$ poiche' il grafo e' connesso

Shortest Paths Tree

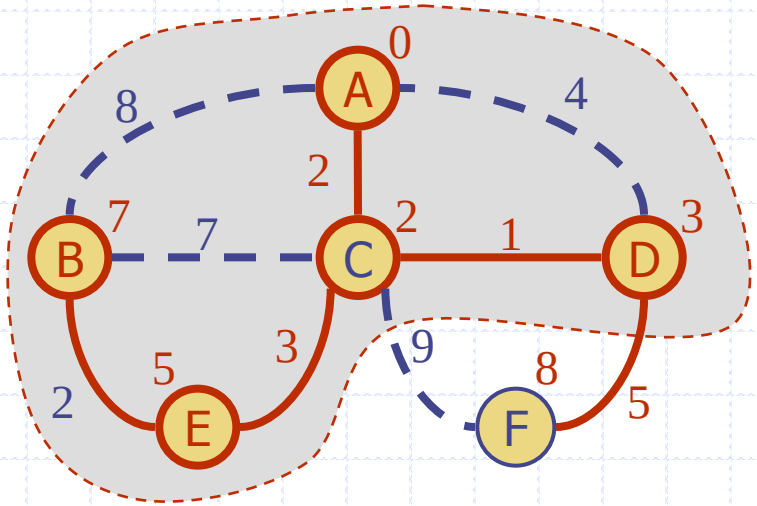
- Usando il metodo template pattern, possiamo estendere l'algoritmo di Dijkstra per restituire un albero di cammini minimi dal vertice di partenza a tutti gli altri vertici
- Memorizziamo con ogni vertice una terza etichetta:
  - arco parent nell'albero dei cammini minimi
- Nel passo di rilassamento di un arco, aggiorniamo l'etichetta parent

Algorithm *DijkstraShortestPathsTree*(*G*, *s*)

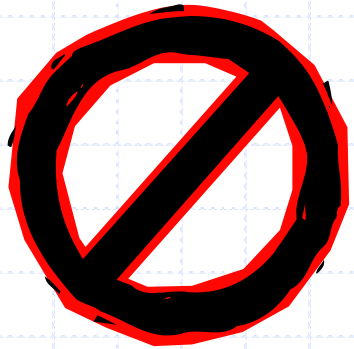
```
…
for all v ∈ G.vertices()
  …
  setParent(v, ∅)
  …
for all e ∈ G.incidentEdges(u)
  { relax edge e }
  z ← G.opposite(u,e)
  r ← getDistance(u) + weight(e)
  if r < getDistance(z)
    setDistance(z,r)
    setParent(z,e)
    Q.replaceKey(getLocator(z),r)
```

Perche' Dijkstra Funziona

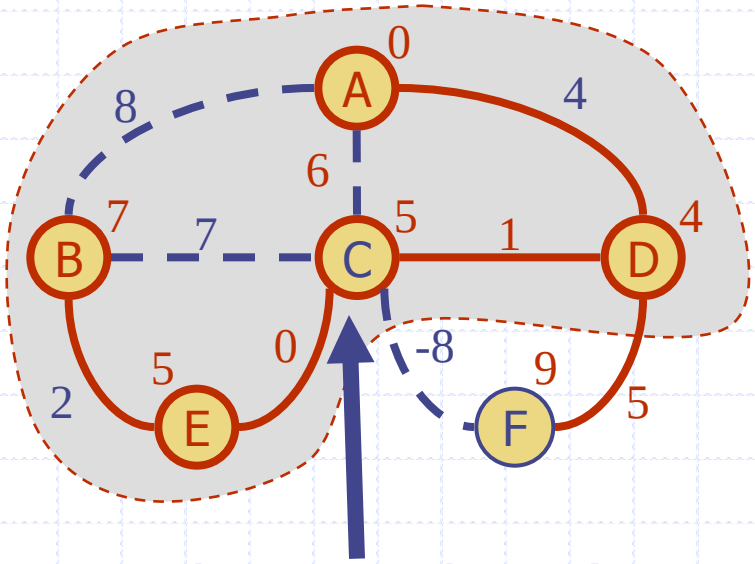
- L'algoritmo di Dijkstra e' basato sull'utilizzo del metodo greedy. Aggiunge vertice di distanza crescente dall'origine.
  - Si assuma che non trovi tutte le distanze minime. Sia F il primo vertice con distanza sbagliata.
  - Quando il nodo precedente, D, su un cammino minimo corretto e' stato considerato, la sua distanza era corretta.
  - Ma l'arco (D,F) e' stato **rilassato** in quel momento!
  - Quindi, finche' d(F)≥d(D), la distanza di F non puo' essere sbagliata.



Perche' non Funziona per Archi con Peso Negativo



- L'algoritmo di Dijkstra e' basato sul metodo greedy. Aggiunge vertici a distanza crescente.
  - Se un nodo con un arco incidente con peso negativo fosse aggiunto piu' tardi nella nuvola, potrebbe alterare le distanze dei vertici gia' nella nuvola
  - L'algoritmo di Bellman-Ford funziona per Grafi con archi di peso negativo ma, attenzione, il problema non e' definito se esiste un ciclo con peso complessivo negativo.



La vera distanza di C e' 1, ma e' gia' nella nuvola quando d(C)=5!