

SPF Routing

Shortest Path First Routing



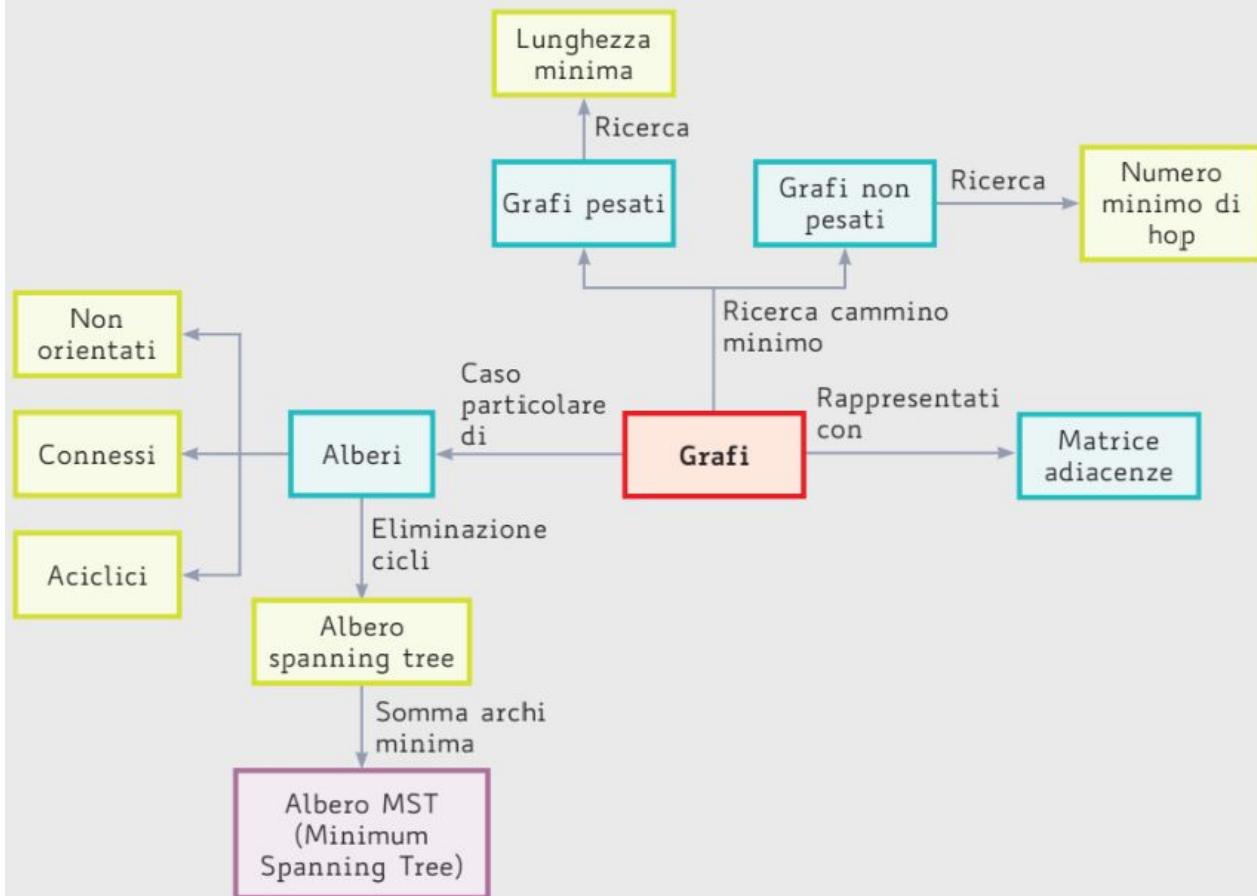
Spanning-Tree

Reti, grafi e alberi

MAPPA CONCETTUALE



didattica inclusiva



Noi chiameremo **router origine** (o di primo hop) il router di default dell'host mittente e **router destinazione** il router predefinito a cui è connesso l'host di destinazione. Il problema da risolvere, che è quello di instradare un pacchetto tra host di origine e host di destinazione, si riconduce chiaramente al problema d'instradare il pacchetto tra questi due router.



Effettuare l'instradamento di un pacchetto in una rete equivale a individuare un "percorso" tra sorgente e destinazione: inoltre il cammino ricercato deve essere il più corto possibile, cioè siamo alla ricerca di un **cammino minimo**.

Concettualmente possiamo indentificare una rete di calcolatori con una struttura dinamica informatica (o matematica) particolare, il **grafo** e, quindi, effettuare la ricerca del cammino minimo tra due router equivale a effettuare la ricerca di un cammino minimo in un grafo:

- ricerca del cammino a **minima distanza**, cioè minimo numeri di archi (hop), se il grafo è non pesato;
- ricerca del cammino a **minima lunghezza**, nel caso di grafo pesato.

In questa lezione analizzeremo le analogie tra le reti e la teoria matematica dei grafi, riprendendo le definizioni e i concetti generali per poi applicarli ai protocolli di routing.



Come vedremo in seguito, il peso di un grafo rappresenta "la distanza" presente tra due nodi e la lunghezza di un percorso si ottiene dalla somma di tali valori: se non viene esplicitamente indicato, si assume che ogni distanza tra due router abbia valore unitario e quindi la distanza tra origine e destinatario è semplicemente il numero di archi intermedi.

AREA DIGITALE

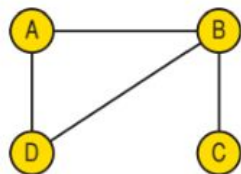


Ripassiamo i grafi

Rappresentazione dei grafi

Tra le possibili rappresentazioni dei grafi ricordiamo la **matrice delle adiacenze**: essendo la matrice una struttura con dimensionamento statico, questa rappresentazione si utilizza nel caso in cui il grafo si riferisca a situazioni dove il numero dei vertici rimane pressoché invariato.

Nella **matrice delle adiacenze** viene riportato, sia in ascissa sia in ordinata, l'elenco dei vertici: nelle celle di incrocio tra due nodi si indica con 1 la presenza di connessione e con 0 l'assenza. Osserviamo che, se il grafo è non orientato, la matrice presenta una simmetria diagonale.



	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	0
D	1	1	0	0

MATRICE DELLE ADIACENZE

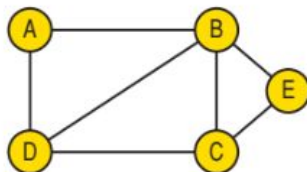
Un grafo può essere rappresentato dalla sua matrice di adiacenza A , di dimensioni $|V| \times |V|$,

il cui generico elemento a_{ij} è definito nel modo seguente

– $a_{ij} = 1$ se $(i,j) \in E$

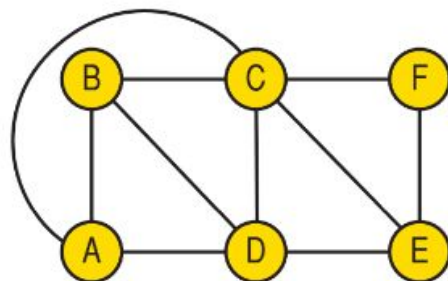
– $a_{ij} = 0$ altrimenti

Vediamo un secondo esempio:



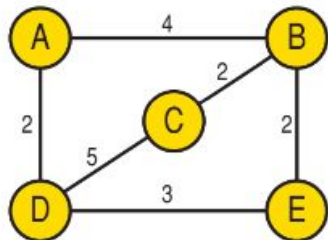
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	1
C	0	1	0	1	1
D	1	1	1	0	0
E	0	1	1	0	0

Vediamo un terzo esempio:

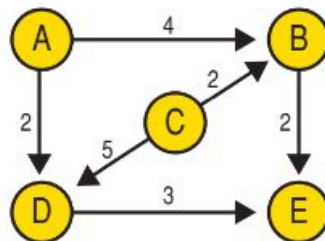


	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	1	1	0	0
C	1	1	0	1	1	1
D	1	1	1	0	1	0
E	0	0	1	1	0	1
F	0	0	1	0	1	0

L'utilizzo delle matrici si presta alla rappresentazione dei grafi pesati: basta sostituire nelle celle della matrice delle adiacenze al valore 1 il valore del peso tra il vertice rappresentato in ordinata e il vertice adiacente in ascissa, indicando con 0 o ∞ l'assenza di connessione.



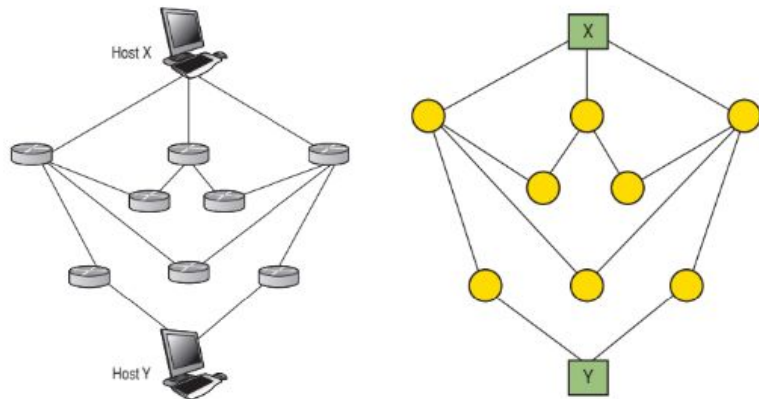
	A	B	C	D	E
A	0	4	∞	2	∞
B	4	0	2	∞	2
C	∞	2	0	5	∞
D	2	∞	5	0	3
E	∞	2	∞	3	0



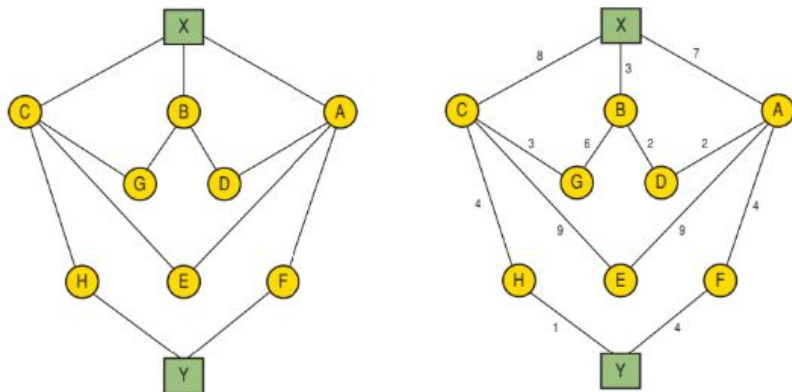
	A	B	C	D	E
A	0	4	∞	2	∞
B	∞	0	∞	∞	2
C	∞	2	0	5	∞
D	∞	∞	∞	0	3
E	∞	∞	∞	∞	0

Grafi e reti

Vediamo ora come utilizzare i grafi per rappresentare le reti. Supponiamo di dover trasmettere un messaggio dall'host X verso l'host Y che appartengono alla rete rappresentata nella figura:

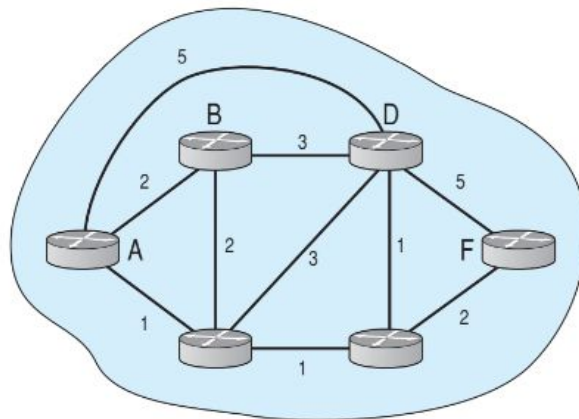
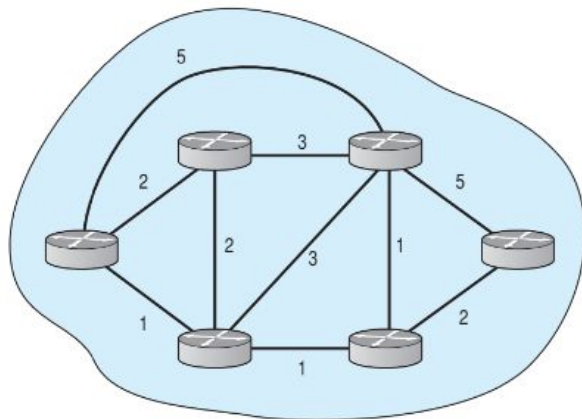


Come primo passaggio assegniamo un identificatore a ogni router e quindi successivamente li indichiamo come nodi e li colleghiamo tra loro come nella seguente figura.



A questo punto aggiungiamo i pesi agli archi e costruiamo la matrice delle adiacenze, dove il peso tra due router è il tempo (oppure il ritardo) per trasmettere un pacchetto tra di essi.

Vediamo un secondo esempio. Alla rete rappresentata nella figura assegniamo una etichetta a ogni router:



e quindi la tabella delle adiacenze.

	A	B	C	D	E	F
A	0	2	1	5	∞	∞
B	2	0	2	3	∞	∞
C	1	2	0	3	1	∞
D	5	3	3	0	1	5
E	∞	∞	1	1	0	2
F	∞	∞	∞	5	2	0

Ricerca del percorso minimo

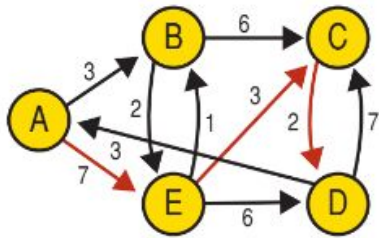
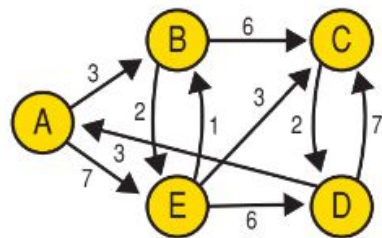
Sono molteplici gli algoritmi che effettuano la ricerca del percorso tra due nodi di un grafo così come gli algoritmi che ricercano i percorsi ottimi, cioè quelli con lunghezza minima (**Shortest Path SP**).

SHORTEST PATH

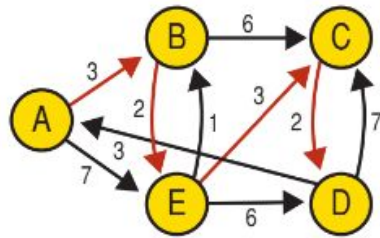
Uno **shortest path** (cammino minimo) dal nodo u al nodo v di V

è un cammino $p = (u, v_1, v_2, \dots, v)$ tale che $w(p)$ è minimo.

Nel seguente esempio individuiamo su un grafo pesato orientato due cammini tra il nodo A e il nodo D.




$p_1(A \dots D) = (A, E, C, D)$
Peso di $p_1 = 7 + 3 + 2 = 12$



$p_2(A \dots D) = (A, B, E, C, D)$
Peso di $p_2 = 3 + 2 + 3 + 2 = 10$

In questo caso il cammino p_2 è il cammino minimo.

Per ogni destinazione è possibile costruire un percorso minimo e l'insieme dei percorsi minimi definisce quello che prende nome di **albero d'inoltro** (forwarding tree) o **albero di consegna** (delivery tree) dove il router sorgente assume il ruolo della radice e gli ultimi router di ogni percorso, quelli che hanno sul loro segmento gli host destinazione, prendono il nome di **router foglia** (leaf router).

L'algoritmo di **Dijkstra** permette di trovare i cammini minimi (**Shortest Paths** ) in un grafo ciclico con pesi non negativi sugli archi: in particolare l'algoritmo può essere utilizzato *parzialmente* per trovare il cammino minimo che unisce due nodi del grafo, *totalmente* per trovare quelli che uniscono un nodo d'origine a tutti gli altri nodi e, ripetendolo più volte, per trovare tutti i cammini minimi da ogni nodo a ogni altro nodo.



Shortest path

The **shortest path** is a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

L'algoritmo di **Bellman-Ford** calcola i cammini minimi di un'unica sorgente su un grafo diretto pesato (dove alcuni pesi degli archi possono anche essere negativi).

Gli algoritmi che permettono di trovare i cammini minimi sono l'algoritmo di Dijkstra e di Bellman-Ford. Bellman-Ford lo abbiamo visto in opera nell'algoritmo Distance Vector. Adesso studiamo Dijkstra che sarà alla base di Link-State e quindi di OSPF.

ALBERO

Un **albero** (tree) è un grafo non orientato, connesso e aciclico.

In un albero non possono esistere quindi percorsi chiusi (cicli) e per ogni coppia di nodi esiste uno e un solo cammino che li congiunge.

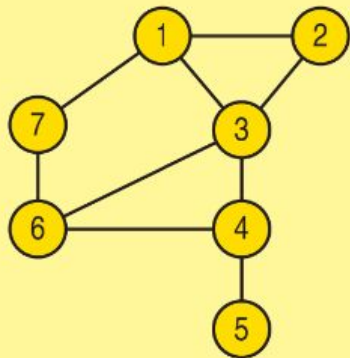
È possibile trasformare un grafo in un albero eliminando gli archi che determinano i cicli: in questo caso si ottiene un sottografo chiamato **albero di ricoprimento** (spanning tree).

ALBERO DI RICOPRIMENTO

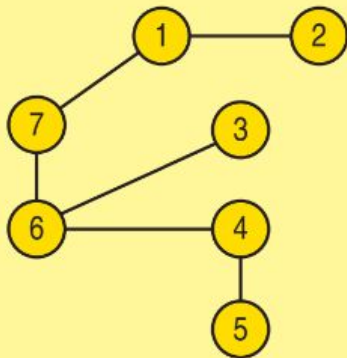
Dato un grafo connesso non orientato $G=(V,E)$, uno **spanning tree** 🇬🇧

(albero di ricoprimento) di G è un sottografo $G'=(V',T)$ tale che:

- G' è un albero
- $V'=V$



Grafo G



Spanning tree di G

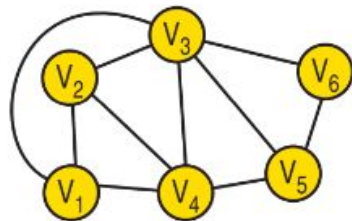


Spanning tree

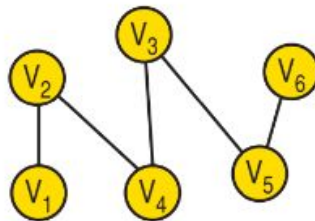
A **spanning tree** T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G . Informally, a spanning tree of G is a selection of edges of G that form a tree spanning every vertex.



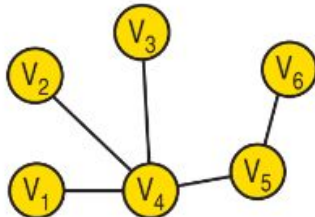
A partire da un grafo è possibile individuare molti alberi ricoprenti e all'aumentare del numero di cicli aumentano anche le differenti combinazioni:



Grafo G



Spanning tree 1



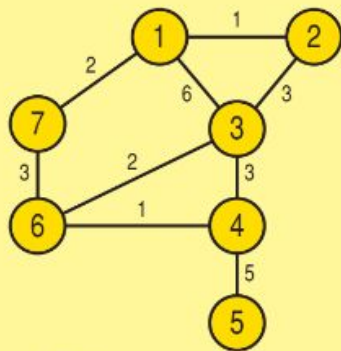
Spanning tree 2

Tra tutti gli alberi ricoprenti individuiamo il **Minimum Spanning Tree (MST)** come l'albero avente somma degli archi minima.

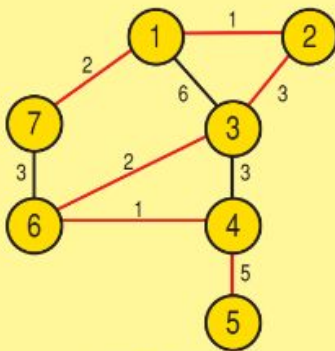
MINIMUM SPANNING TREE

Dato un grafo pesato connesso non orientato $G = (V, E)$, un **minimum spanning tree** (albero di ricoprimento minimo) di G è uno **spanning tree** $G' = (V, T)$ di peso minimo, cioè tale che risulti minimo $w(T)$ dove:

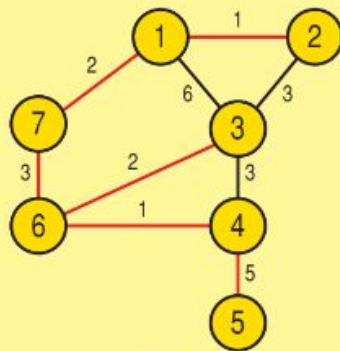
$$w(T) = \sum_{(i,j) \in T} w(i,j)$$



Grafo pesato G



Un MST di G
 $w(T) = 14$ ~~36~~



Un altro MST di G
 $w(T) = 14$ ~~36~~

Come è stato riportato nella figura precedente per lo stesso grafo possono esistere diversi **MST**, cioè **Spanning Tree** aventi tutti lo stesso valore (minimo) come somma degli archi.

peso totale del grafo sx:

- da 1 a 2 : 1
- da 1 a 3 : 4
- da 1 a 4 : 7
- da 1 a 5 : 13
- da 1 a 6 : 6
- da 1 a 7 : 2

peso totale : 36


peso totale del grafo dx:

- da 1 a 2 : 1
- da 1 a 3 : 7
- da 1 a 4 : 6
- da 1 a 5 : 11
- da 1 a 6 : 6
- da 1 a 7 : 5

peso totale : 36

Come è stato riportato nella figura precedente per lo stesso grafo possono esistere diversi **MST**, cioè **Spanning Tree** aventi tutti lo stesso valore (minimo) come somma degli archi.

SPANNING TREE OTTIMO

Definiamo **Spanning Tree Ottimo** lo **Spanning Tree** formato dai cammini a costo minimo (**shortest path**) che uniscono il nodo radice agli altri nodi del grafo (prende anche il nome di **sink tree** .

L'individuazione dei **sink tree** è l'obiettivo di un algoritmo di instradamento in un router e noi analizzeremo a tale scopo, nelle prossime lezioni, l'algoritmo di **Dijkstra** e di **Bellman-Ford**.



Sink tree

Sink tree (minimum spanning tree): set of all optimal paths from all sources to a given destination. Can be found using the shortest path algorithm from destination to all sources giving optimal though not necessarily unique route.

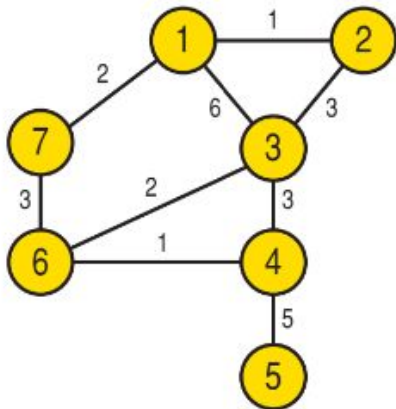
ESERCIZI

3 Una rete di 5 nodi è rappresentata dalla seguente tabella delle adiacenze.

	V1	V2	V3	V4	V5
V1	0	10	5	∞	∞
V2	10	0	3	10	5
V3	5	3	0	∞	∞
V4	∞	10	∞	0	3
V5	∞	5	∞	3	0

Si disegni il grafo non orientato pesato corrispondente alla rete.

6 Data la rete rappresentata nella figura si determini la tabella delle adiacenze considerando un grafo non orientato pesato. I collegamenti sono tutti bidirezionali e la loro capacità è indicata nella figura.



Dijkstra

algoritmo per trovare il percorso a costo minore in un grafo pesato

Dijkstra's algorithm

🌐 49 languages ▾

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

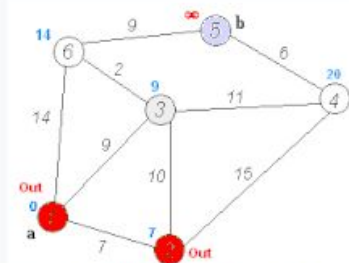
From Wikipedia, the free encyclopedia

Not to be confused with [Dykstra's projection algorithm](#).

Dijkstra's algorithm (/ˈdaɪkstrəz/ *DYKE-strəz*) is an [algorithm](#) for finding the [shortest paths](#) between [nodes](#) in a weighted [graph](#), which may represent, for example, a [road network](#). It was conceived by [computer scientist](#) [Edsger W. Dijkstra](#) in 1956 and published three years later.^{[4][5][6]}

Dijkstra's algorithm finds the shortest path from a given source node to every other node.^{[7]:196–206} It can be used to find the shortest path to a specific destination node, by terminating the algorithm after determining the shortest path to the destination node. For example, if the nodes of the graph represent cities, and the costs of edges represent the average distances between pairs of cities connected by a direct road, then Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. A common application of shortest path algorithms is network [routing protocols](#), most notably [IS-IS](#) (Intermediate System to Intermediate System) and [OSPF](#) (Open Shortest Path First). It is also employed as a [subroutine](#) in algorithms such as [Johnson's](#)

Dijkstra's algorithm



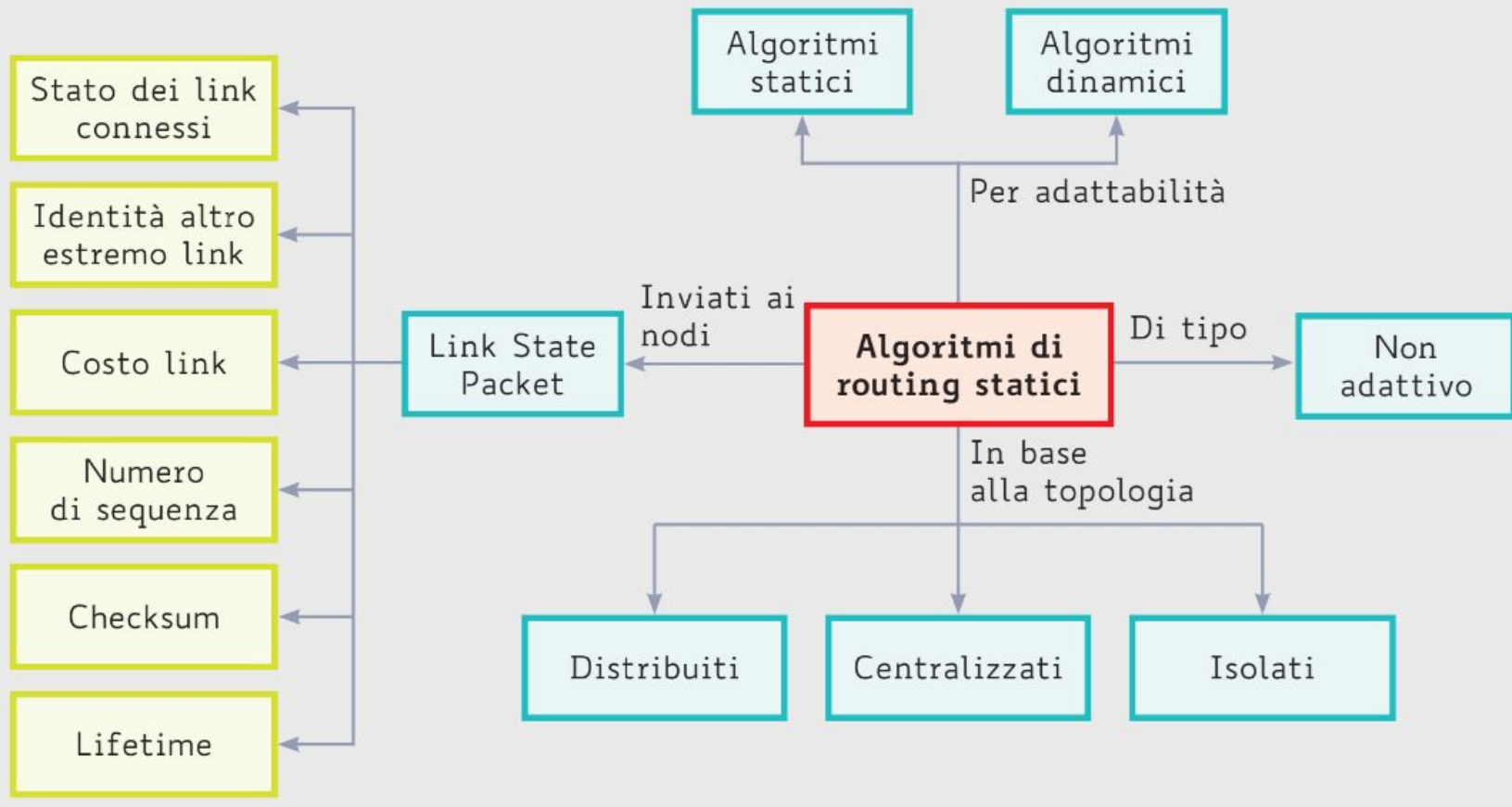
Dijkstra's algorithm to find the shortest path between *a* and *b*. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with

algorithmo animato https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
pronuncia https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

MAPPA CONCETTUALE



didattica inclusiva



L'algoritmo di Dijkstra

L'algoritmo di Dijkstra è un algoritmo che rientra tra gli **shortest path routing** e permette di calcolare l'**albero dei cammini minimi** tra un nodo di un grafo e tutti gli altri in modo da configurare le tabelle di routing: data la sua natura statica, se cambia la configurazione della rete, è necessario ricalcolare l'albero dei cammini minimi ripartendo da capo.

È un algoritmo di tipo **centralizzato** in quanto ciascun nodo calcola il cammino ottimo da se stesso verso tutti gli altri nodi in modo indipendente, partendo unicamente dalle informazioni complete sulla topologia della rete e sulle caratteristiche dei link.

Prima di vedere come procedere, definiamo innanzitutto l'obiettivo che è quello di **trovare i cammini minimi tra un nodo 1 (sorgente) e tutti gli altri nodi** partendo dall'ipotesi che tutti gli archi abbiano pesi positivi.

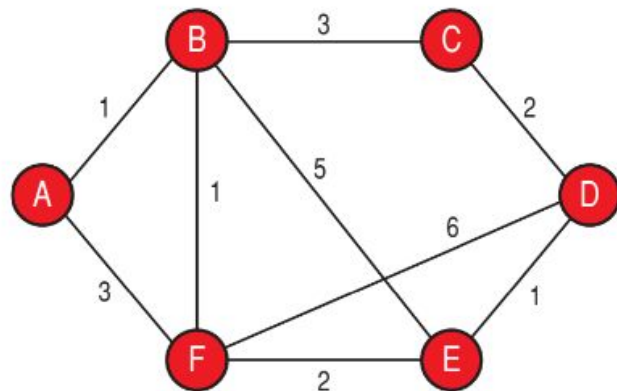
Il metodo di **Dijkstra** si assegna ai nodi delle etichette, che possono essere:

- **temporanee**: il costo massimo per la raggiungibilità del nodo in esame;
- **permanenti**: costo del cammino minimo.

L'algoritmo di calcolo modifica le etichette temporanee cercando di minimizzarle e di renderle permanenti.

ESEMPIO

Vediamo un semplice esempio applicando l'algoritmo al **grafo non orientato** della figura:



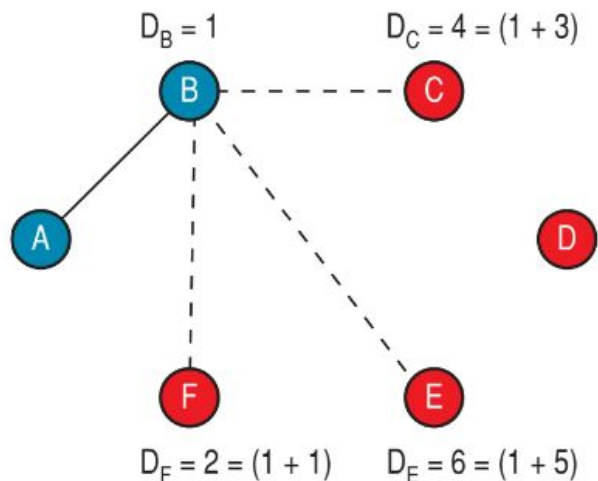
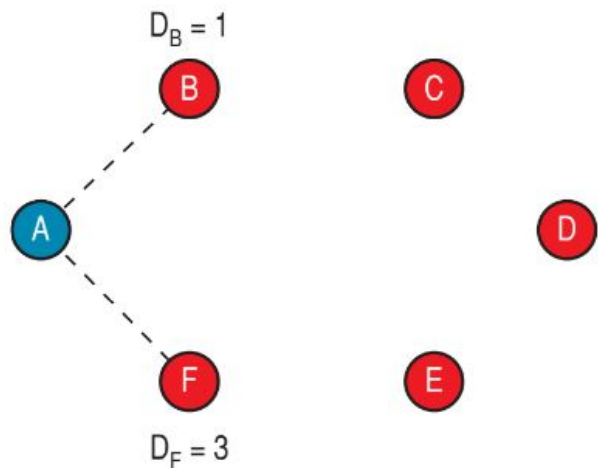
Scegliamo “a piacere” un nodo, in quanto il risultato che otteniamo è ininfluente dal nodo di partenza, e iniziamo a scrivere le **etichette provvisorie** sui due nodi a esso adiacenti.

Partiamo da A ed etichettiamo B con 1 e F con 3.

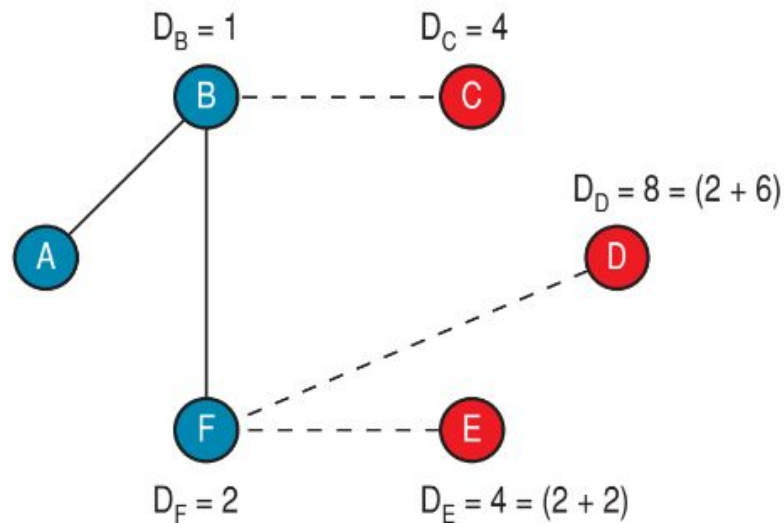
Scegliamo ora il “percorso meno costoso” che si è individuato, cioè quello che parte dal nodo B, e ripetiamo le stesse operazioni assegnando le **etichette provvisorie** ai nodi C, F ed E ottenute come somma dei percorsi dal nodo precedente (1) e dei nuovi pesi dei rispettivi archi:



La scelta secondo la quale il nodo successivo da visitare è quello più vicino a uno dei nodi già visitati dà il nome all'algoritmo **Shortest Path First**, cioè **prima il percorso più breve**.



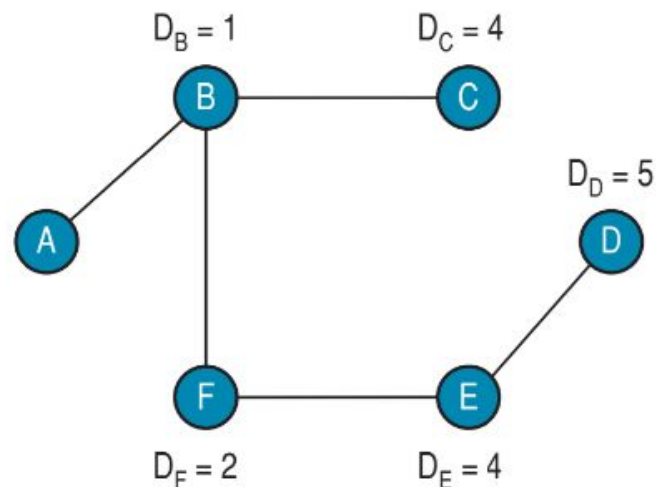
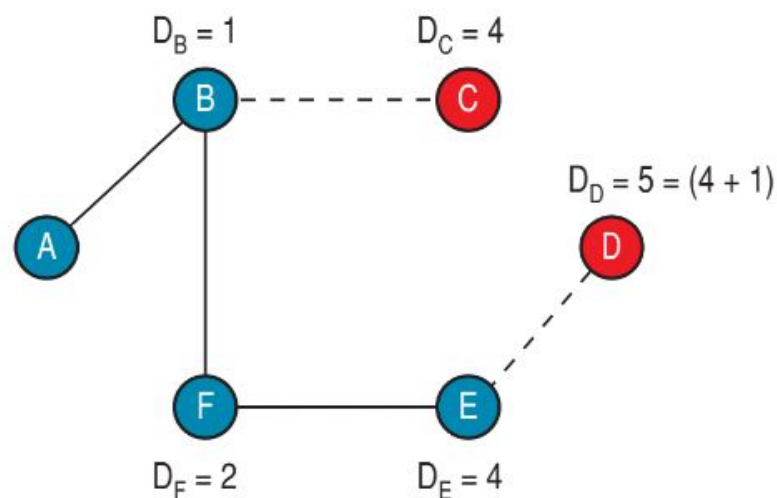
Procediamo in modo analogo scegliendo il percorso più corto tra i tre sino a ora individuati, cioè quello che ha raggiunto il nodo F (peso 2), e aggiorniamo le etichette dei nodi a esso adiacenti ($D_D = 2 + 6 = 8$, $D_E = 2 + 2$):



Con lo stesso criterio il prossimo nodo da analizzare è il nodo E (peso 4): dal nodo E possiamo raggiungere il nodo D con peso $D_D = 4 + 1 = 5$ che è minore del precedente valore dell'etichetta provvisoria: la sostituiamo con una *nuova etichetta provvisoria*.

Il nodo con peso minore tra quelli che non sono ancora stati analizzati è il C che ha peso 4, ma non porta migliorie in quanto la sua distanza da D ha peso 2 che peggiorerebbe l'etichetta parziale.

Ultimo nodo è il nodo D, ma anch'esso non introduce migliorie e quindi le etichette **provvisorie** diventano **permanenti** e si ottiene il seguente **albero minimo**:



ESERCIZI

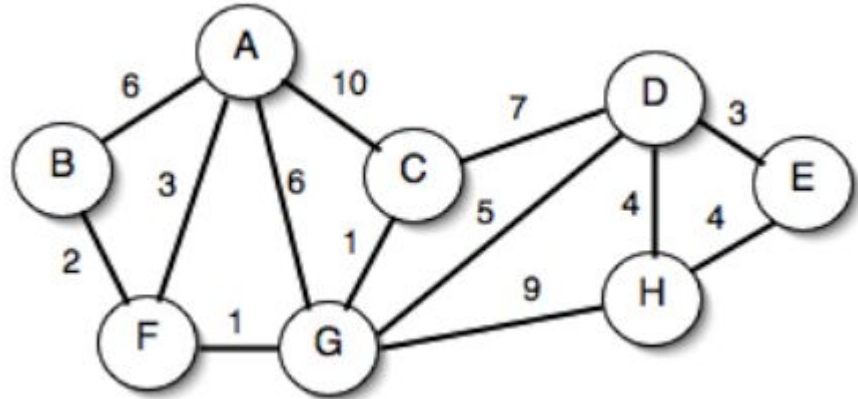
Rispondi alle seguenti domande

1. Che cosa è un grafo ?
2. Per cosa possiamo utilizzare un grafo ?
3. Che cosa si intende per instradamento di un pacchetto ?
4. Che cosa è la matrice delle adiacenze ?
5. Che cosa è il percorso a costo minimo SP Shortest Path ?
6. Che cosa è un albero ?
7. Che cosa è un albero di inoltro ?
8. Che cosa è un albero di ricoprimento SPANNING TREE ?
9. Che cosa è un albero minimo di ricoprimento MINIMUM SPANNING TREE ?
10. A cosa serve l'algoritmo di Dijkstra ?

Risolvi

Dal seguente grafo

1. Determinare due diversi alberi di ricoprimento
2. Determinare il MST partendo da A ed il suo peso



IMPLEMENTAZIONE SOFTWARE di DIJKSTRA



Nei grafi delle figure che seguono i nodi in colore nero sono quelli da visitare, mentre i nodi azzurri sono quelli già visitati: a ciascuno è associato il costo del percorso dal nodo A che è il nodo di partenza. La situazione iniziale è quindi la seguente con il nodo A come nodo corrente (FIGURA 25):

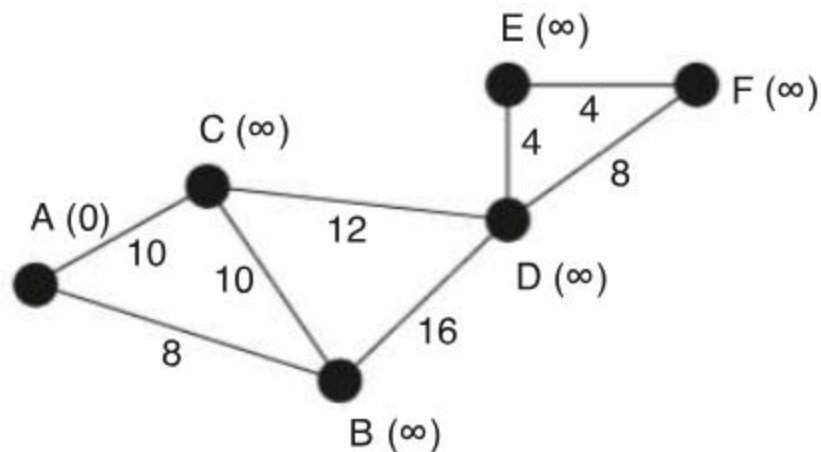


FIGURA 25

La prima iterazione dell'algoritmo prende in considerazione i nodi B e C adiacenti al nodo corrente calcolando rispettivamente il costo 8 ($0 + 8$) per B e il costo 10 ($0 + 10$) per C; il nodo A viene eliminato dall'insieme dei nodi da visitare (FIGURA 26):

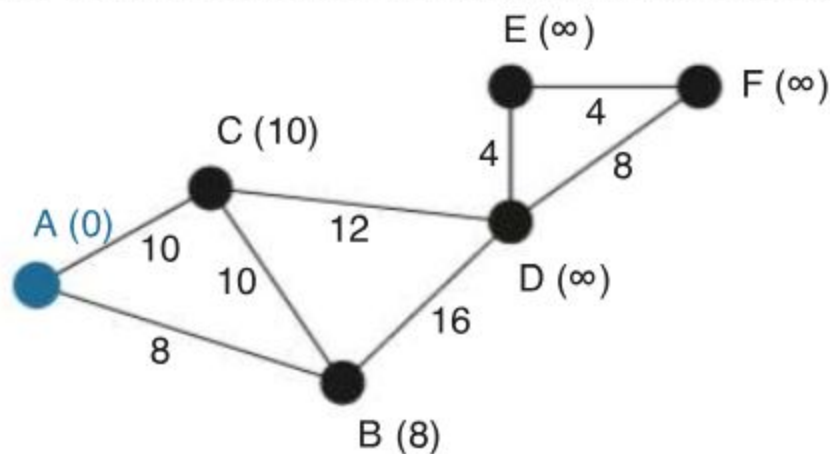


FIGURA 26

Il nuovo nodo corrente è B in quanto ha il costo minimo tra tutti quelli non ancora visitati; i suoi nodi adiacenti non ancora visitati sono C, per il quale viene calcolato un costo di 18 ($8 + 10$) che essendo superiore al costo associato non viene aggiornato, e D per il quale viene calcolato un costo di 24 ($8 + 16$); il nodo B viene eliminato dall'insieme dei nodi da visitare (FIGURA 27):

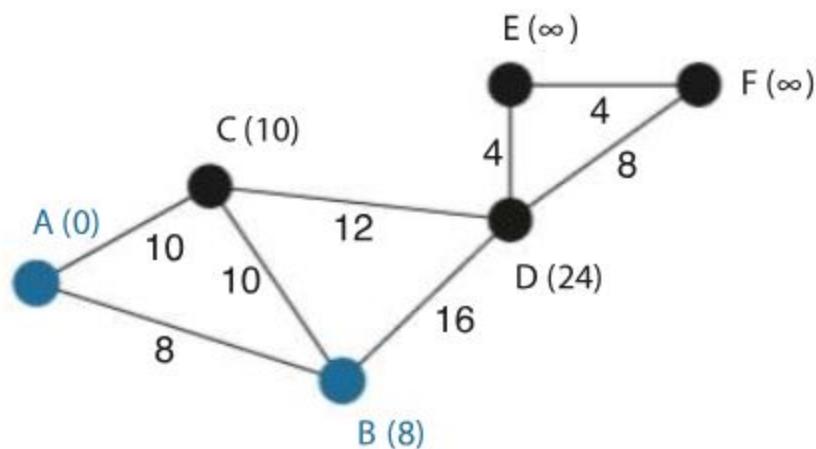


FIGURA 27

Il nuovo nodo corrente è C in quanto ha il costo minimo tra tutti quelli non ancora visitati; l'unico nodo adiacente non ancora visitato è D, per il quale viene calcolato un costo di 22 ($10 + 12$) che, essendo inferiore al costo associato, viene aggiornato; il nodo C viene eliminato dall'insieme dei nodi da visitare (FIGURA 28):

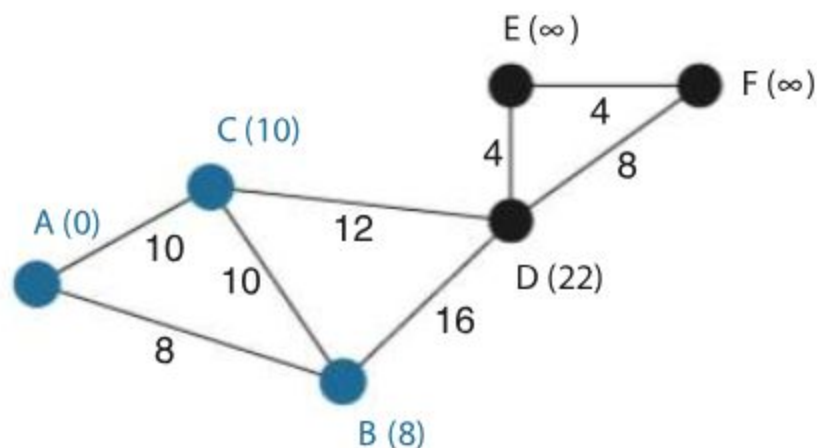


FIGURA 28

Il nuovo nodo corrente è D in quanto ha il costo minimo tra tutti quelli non ancora visitati; i suoi nodi adiacenti non ancora visitati sono E, per il quale viene calcolato un costo di 26 ($22 + 4$), ed F per il quale viene calcolato un costo di 30 ($22 + 8$); entrambi i costi vengono aggiornati e il nodo D viene eliminato dall'insieme dei nodi da visitare (FIGURA 29):

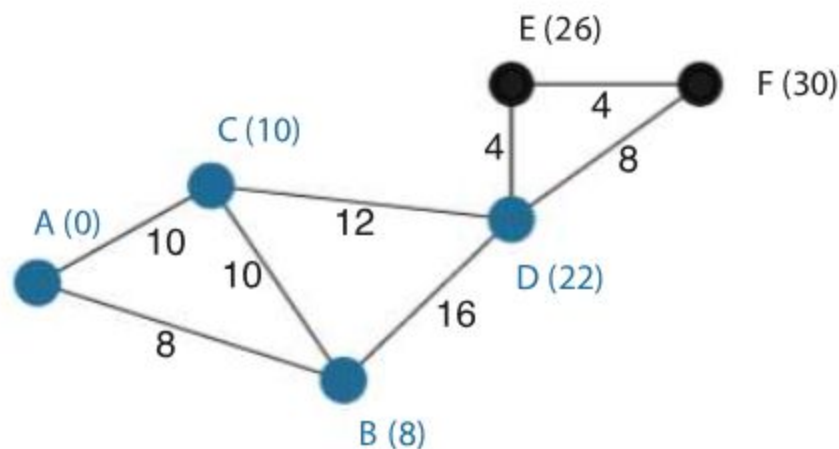


FIGURA 29

Il nuovo nodo corrente è E in quanto ha il costo minimo tra tutti quelli non ancora visitati; l'unico nodo adiacente non ancora visitato è F, per il quale viene calcolato un costo di 30 ($26 + 4$) che, non essendo inferiore al costo associato, non viene aggiornato; il nodo E viene eliminato dall'insieme dei nodi da visitare (FIGURA 30):

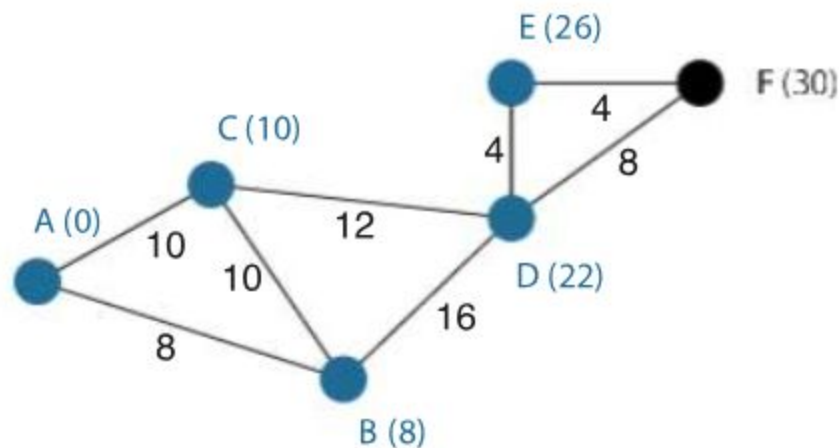


FIGURA 30

Il nuovo nodo corrente è F, unico nodo non ancora visitato e privo di nodi adiacenti non ancora visitati; l'algoritmo termina eliminando il nodo F dall'insieme dei nodi da visitare avendo determinato i costi dei percorsi ottimi dal nodo di partenza A a tutti gli altri nodi del grafo (FIGURA 31):

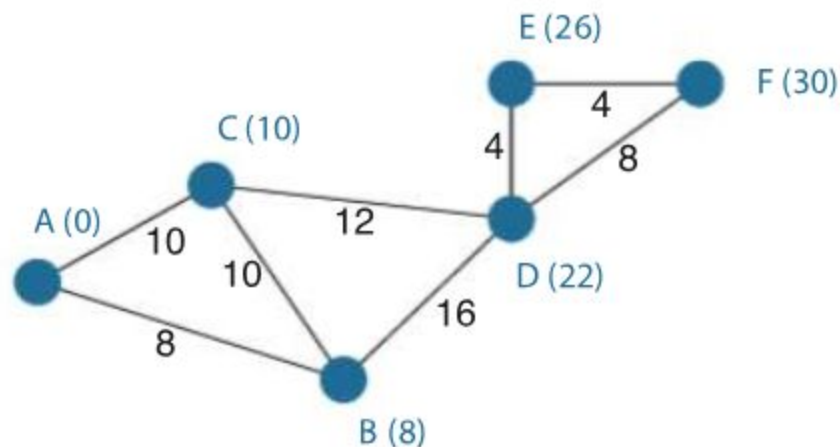


FIGURA 31