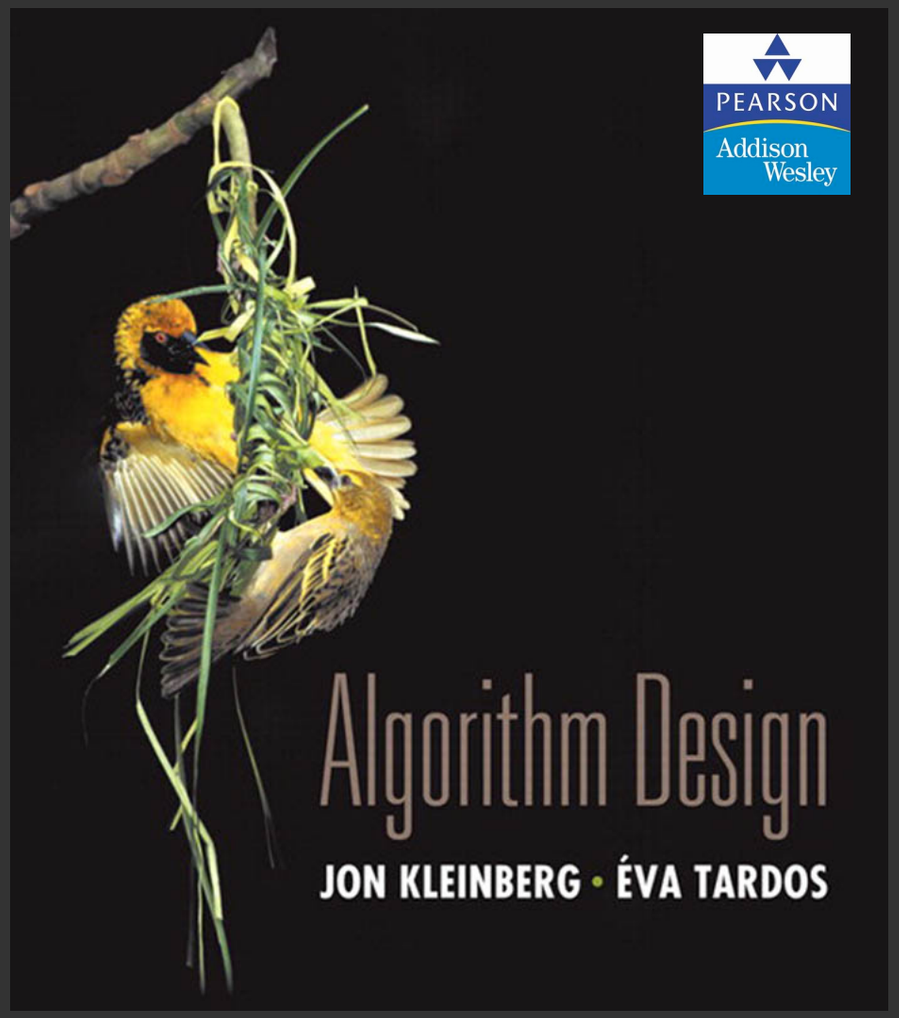


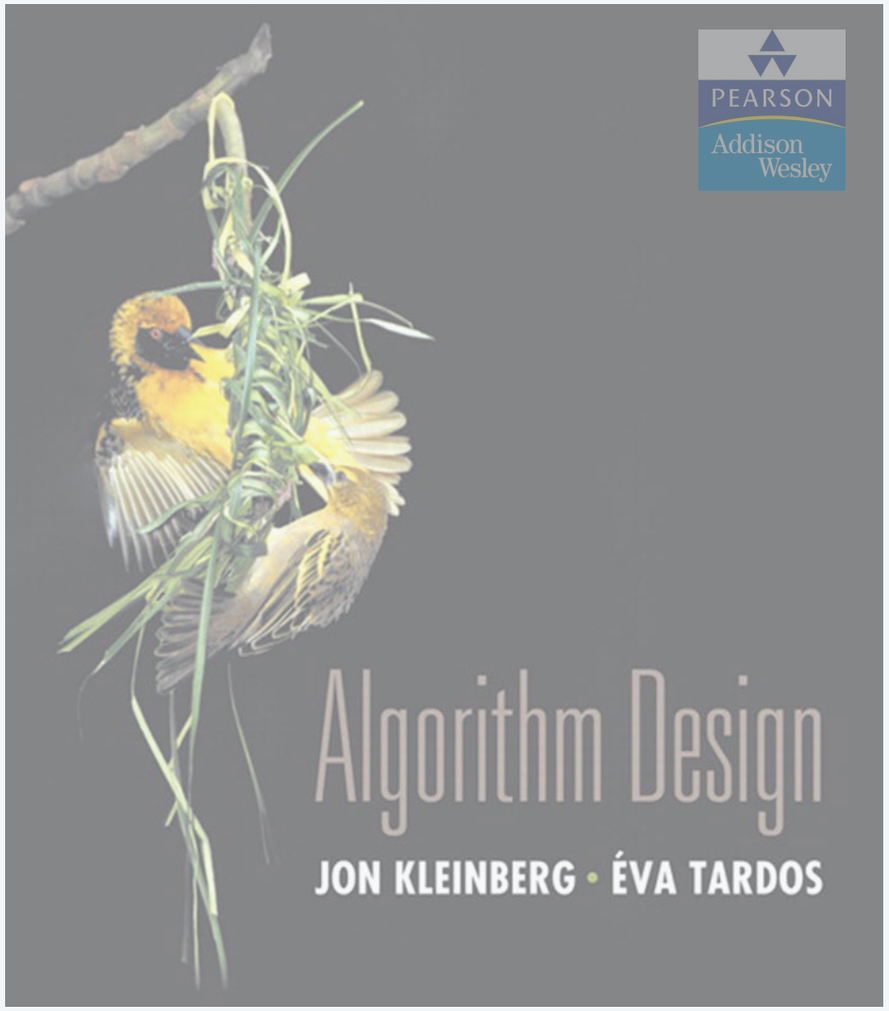
4. Algoritmi Avidi II

- *algoritmo di Dijkstra*
- *minimo albero ricoprente*
- *Prim, Kruskal*
- *clustering a massima spaziatura*

Traduzione e adattamento di Vincenzo Bonifaci
Original lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 10/06/2020 11:26



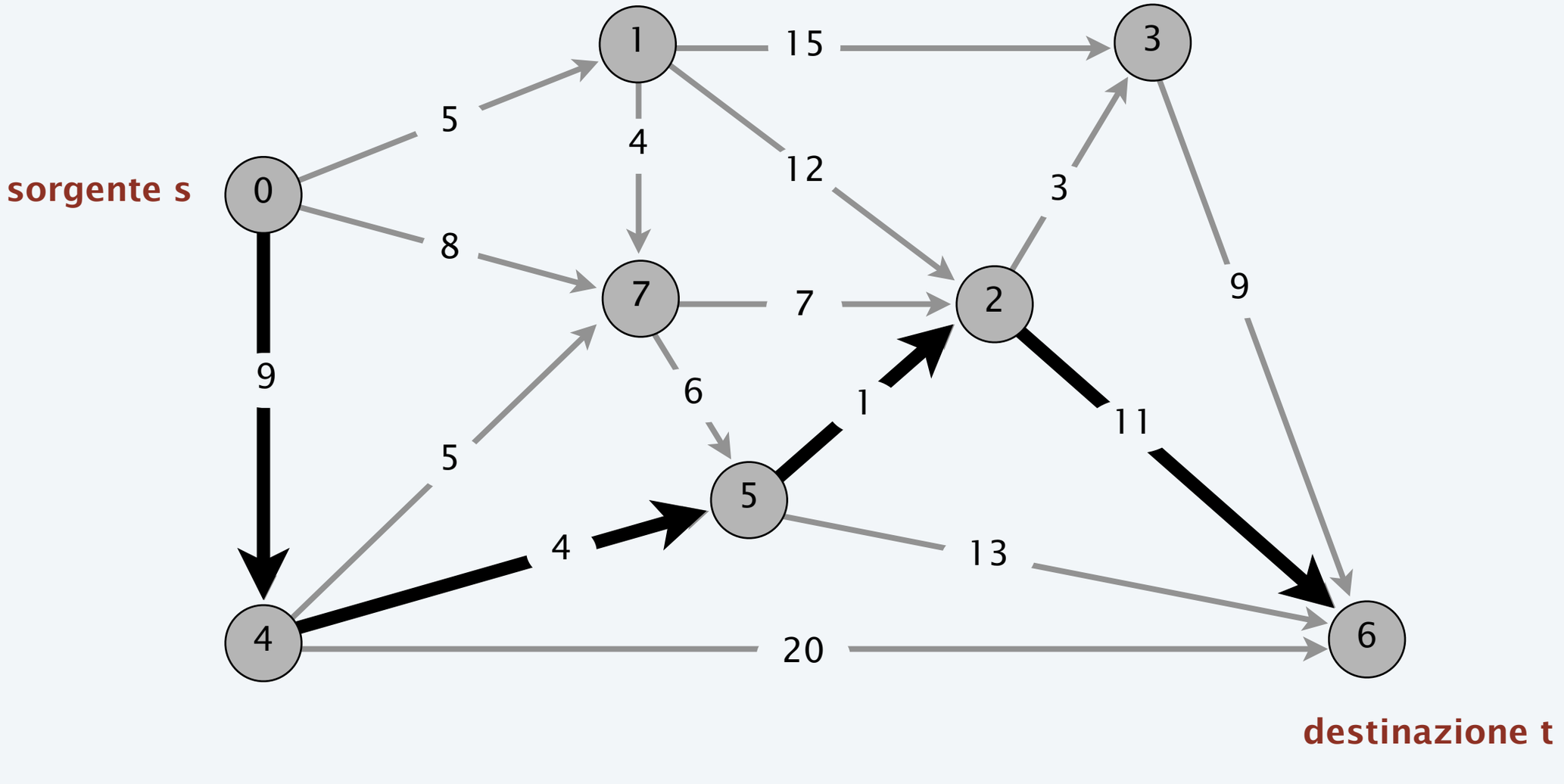
SECTION 4.4

4. ALGORITMI AVIDI II

- *algoritmo di Dijkstra*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Problema del cammino minimo tra un'unica coppia

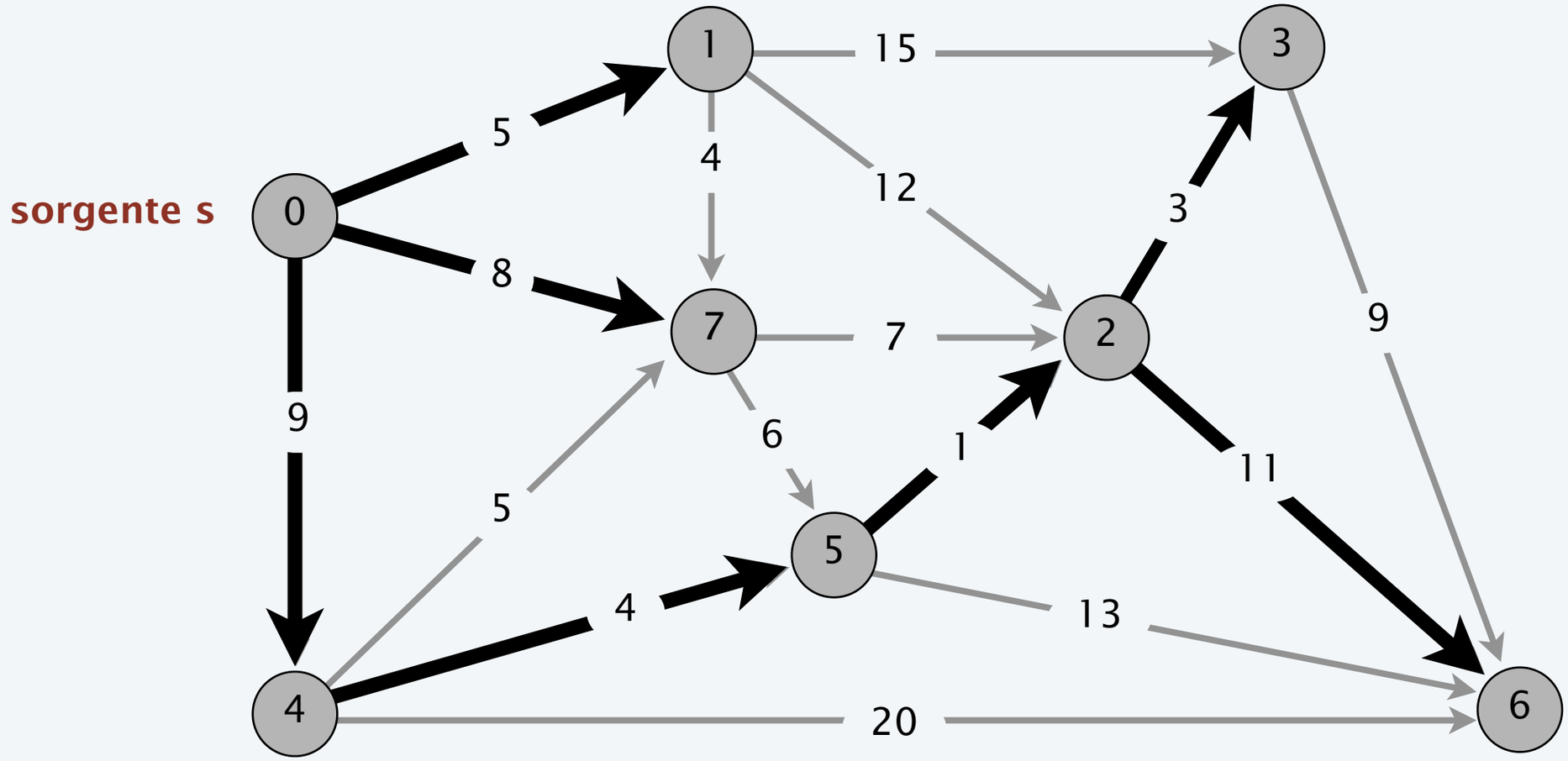
Problema. Dato un digrafo $G = (V, E)$, lunghezze degli archi $\ell_e \geq 0$, sorgente $s \in V$, e destinazione $t \in V$, trovare un cammino orientato di lunghezza minima da s a t.



lunghezza del cammino = 9 + 4 + 1 + 11 = 25

Problema dei cammini minimi da un'unica sorgente

Problema. Dato un digrafo $G = (V, E)$, lunghezze degli archi $\ell_e \geq 0$, sorgente $s \in V$, trovare un cammino orientato di lunghezza minima da s ad ogni altro nodo.



albero dei cammini minimi

Cammini minimi: quiz 1

Supponiamo di modificare la lunghezza di ogni arco di G in uno dei modi seguenti. Per quale di questi è vero che ogni cammino minimo di G è un cammino minimo del grafo modificato G'?

A. Aggiungere 17.

B. Moltiplicare per 17.

C. Sia A che B.

D. Né A né B.

Cammini minimi: quiz 2

Quale variante è usata nei GPS delle automobili?

A. Unica sorgente: da un nodo s ad ogni altro nodo.

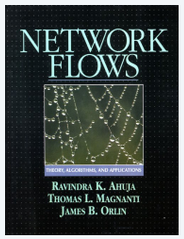
B. Unica destinazione: da ogni altro nodo ad un nodo t.

C. Sorgente-destinazione: da un nodo s ad un altro nodo t.

D. Tutte le coppie: tra tutte le coppie di nodi.

Applicazioni dei cammini minimi

- Diagrammi PERT/CPM per la gestione di progetti.
- Instradamento geografico.
- Elaborazione di immagini (algoritmo seam carving).
- Navigazione robotica.
- Mappatura di texture.
- Tipografia in LaTeX.
- Pianificazione del traffico urbano.
- Instradamento di messaggi di telecomunicazioni.
- Protocolli di instradamento di rete (OSPF, BGP, RIP).
- Instradamento ottimo di veicoli in un pattern di congestione dato.



Network Flows: Theory, Algorithms, and Applications, by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

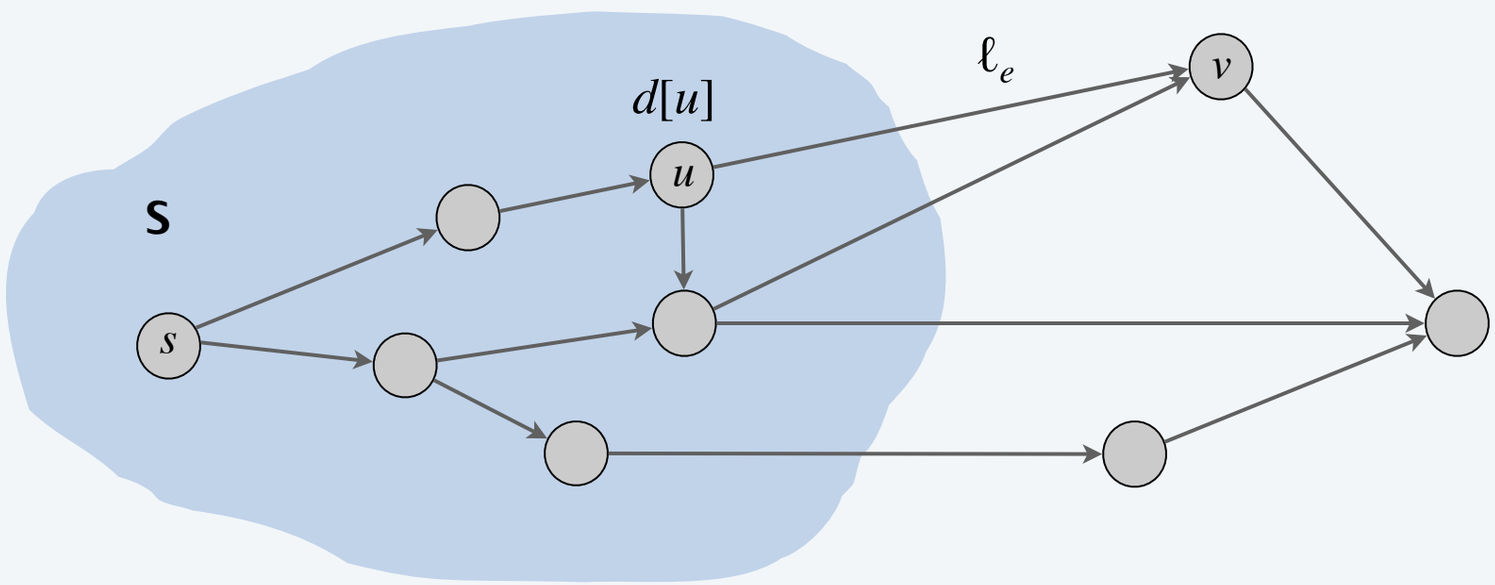
Algoritmo di Dijkstra (cammini minimi da un'unica sorgente)

Approccio avido. Teniamo un insieme di nodi esplorati S per i quali l'algoritmo ha determinato $d[u]$ = lunghezza di un cammino minimo $s \rightsquigarrow u$.

- Inizializza $S \leftarrow \{ s \}$, $d[s] \leftarrow 0$.
- Scegli iterativamente un nodo inesplorato $v \notin S$ che minimizzi

$$\pi(v) \;=\; \min_{e=(u,v)\,:\,u \in S} d[u] + \ell_e$$

lunghezza minima di un cammino da s a qualche nodo u della zona esplorata S, seguito da un singolo arco $e = (u, v)$



Algoritmo di Dijkstra (cammini minimi da un'unica sorgente)

Approccio avido. Teniamo un insieme di nodi esplorati S per i quali l'algoritmo ha determinato $d[u]$ = lunghezza di un cammino minimo $s \rightsquigarrow u$.

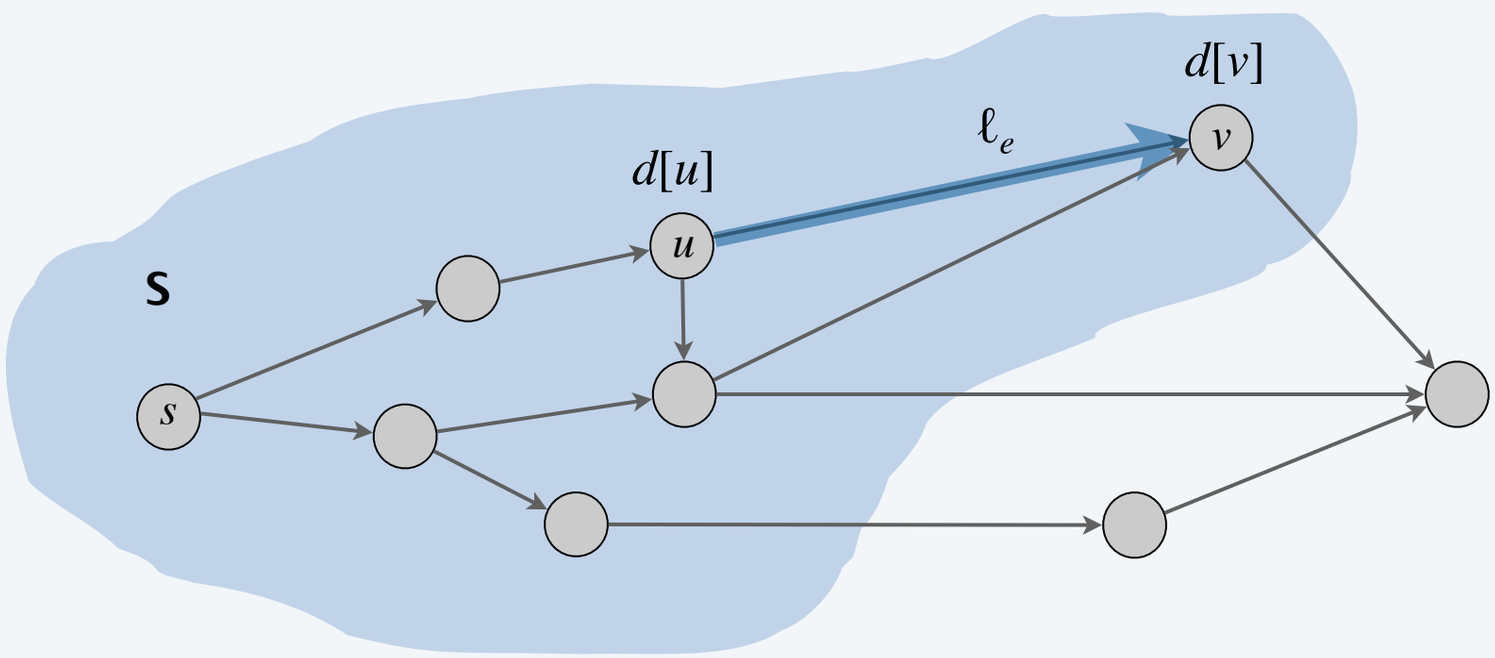
- Inizializza $S \leftarrow \{ s \}$, $d[s] \leftarrow 0$.
- Scegli iterativamente un nodo inesplorato $v \notin S$ che minimizzi

$$\pi(v) \;=\; \min_{e=(u,v)\,:\,u \in S} d[u] + \ell_e$$

lunghezza minima di un cammino da s a qualche nodo u della zona esplorata S, seguito da un singolo arco $e = (u, v)$

Aggiungi v ad S, e assegna $d[v] \leftarrow \pi(v)$.

- Per ricordare il cammino, assegna $pred[v] \leftarrow e$ per l'e che dà il minimo.



Algoritmo di Dijkstra: dimostrazione di correttezza

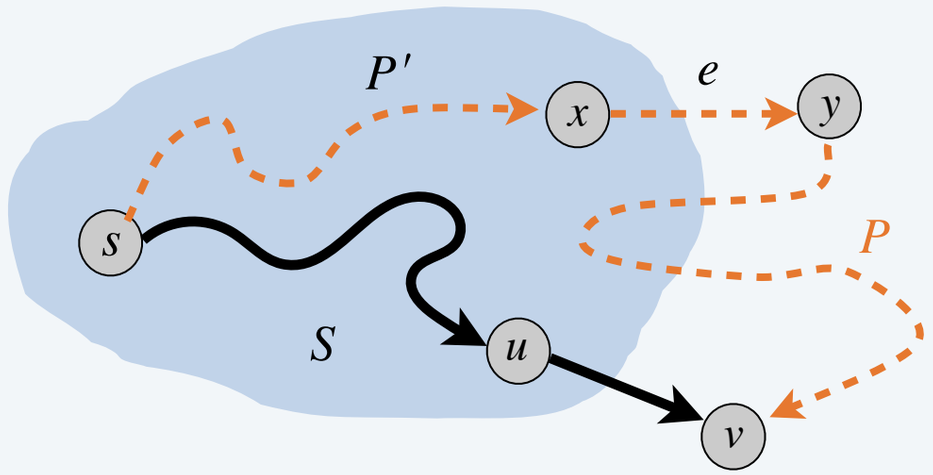
Invariante. Per ogni nodo $u \in S$: $d[u]$ = lunghezza di un cammino minimo $s \rightsquigarrow u$.

Dim. [per induzione su $|S|$]

Caso base: $|S| = 1$ è facile poiché $S = \{ s \}$ e $d[s] = 0$.

Ipotesi induttiva: Assumiamo sia vero per $|S| \geq 1$.

- Sia v il nodo successivo aggiunto ad S, e sia (u, v) l'arco finale.
- Un cammino minimo $s \rightsquigarrow u$ più (u, v) è un cammino $s \rightsquigarrow v$ di lunghezza $\pi(v)$.
- Sia P qualunque altro cammino $s \rightsquigarrow v$. Mostriamo che non è più breve di $\pi(v)$.
- Sia $e = (x, y)$ il primo arco di P che lascia S, e sia P' il sottocammino da s ad x.
- La lunghezza di P è già $\geq \pi(v)$ nel momento in cui raggiunge y:



$$\ell(P) \geq \ell(P') + \ell_e \geq d[x] + \ell_e \geq \pi(y) \geq \pi(v) \quad \blacksquare$$

- $\ell(P) \geq \ell(P') + \ell_e$: lunghezze non-negative
- $\ell(P') + \ell_e \geq d[x] + \ell_e$: ipotesi induttiva
- $d[x] + \ell_e \geq \pi(y)$: definizione di $\pi(y)$
- $\pi(y) \geq \pi(v)$: Dijkstra sceglie v invece di y

Algoritmo di Dijkstra: implementazione efficiente

Ottimizzazione critica 1. Per ogni nodo inesplorato $v \notin S$:
memorizziamo esplicitamente $\pi[v]$ invece di calcolarlo dalla definizione

$$\pi(v) \;=\; \min_{e=(u,v)\,:\,u\in S} d[u] + \ell_e$$

- Per ogni $v \notin S$: $\pi(v)$ può solo diminuire (perché l'insieme S cresce).

- In particolare, supponiamo u sia aggiunto ad S e sia scelto un arco $e = (u, v)$ che parte da u. È sufficiente aggiornare:

$$\pi[v] \leftarrow \min \{\, \pi[v],\; \pi[u] + \ell_e) \,\}$$

ricordare: per ogni $u \in S$,
$\pi[u] = d[u]$ = lunghezza del cammino minimo $s \to u$

Ottimizzazione critica 2. Usiamo una coda di priorità (PQ)
per scegliere un nodo inesplorato che minimizzi $\pi[v]$.

Algoritmo di Dijkstra: implementazione efficiente

Implementazione.

- L'algoritmo mantiene $\pi[v]$ per ogni nodo v.
- La coda di priorità mantiene i nodi inesplorati, con $\pi[\cdot]$ come priorità.
- Quando u è rimosso da PQ, $\pi[u]$ = lunghezza di un cammino minimo $s \rightsquigarrow u$.

DIJKSTRA (V, E, ℓ, s)

FOREACH $v \neq s$: $\pi[v] \leftarrow \infty, pred[v] \leftarrow null; \ \pi[s] \leftarrow 0.$

Crea una coda di priorità vuota pq.

FOREACH $v \in V$: INSERT$(pq, v, \pi[v])$.

WHILE (IS-NOT-EMPTY(pq))

  $u \leftarrow$ DEL-MIN(pq).

  FOREACH arco $e = (u, v) \in E$ che parte da u:

    IF $(\pi[v] > \pi[u] + \ell_e)$

      DECREASE-KEY$(pq, v, \pi[u] + \ell_e)$.

      $\pi[v] \leftarrow \pi[u] + \ell_e$; $pred[v] \leftarrow e$.

Algoritmo di Dijkstra: che coda di priorità utilizzare?

Performance. Dipende da PQ: n INSERT, n DELETE-MIN, $\leq m$ DECREASE-KEY.

- Implementazione basata su array è ottima per grafi densi. ← $\Theta(n^2)$ archi
- Heap binario molto più rapido per grafi sparsi. ← $\Theta(n)$ archi
- Heap 4-ario valido in situazioni in cui la performance è critica.

implementazione della coda di priorità	INSERT	DELETE-MIN	DECREASE-KEY	totale
array disordinato	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
heap binario	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
heap d-ario (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
heap di Fibonacci (Fredman–Tarjan 1984)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
coda di priorità intera (Thorup 2004)	$O(1)$	$O(\log \log n)$	$O(1)$	$O(m + n \log \log n)$

assume $m \geq n$ † ammortizzato

Cammini minimi: quiz 3

Come risolvere il problema dei cammini minimi con unica sorgente in grafi non orientati con lunghezze degli archi positive?

A. Sostituire ogni arco non orientato con due archi contrapposti di uguale lunghezza. Lanciare Dijkstra sul digrafo risultante.

B. Modificare l'algoritmo di Dijkstra in modo che quando processa il nodo u, vengano considerati tutti gli archi incidenti ad u (non solo quelli uscenti da u).

C. Sia A che B.

D. Né A né B.

Cammini minimi: quiz 3

Teorema. [Thorup 1999] Si può risolvere il problema dei cammini minimi da un'unica sorgente in grafi non orientati con lunghezze positive intere in tempo $O(m)$.

Nota. Non esplora i nodi in ordine di distanza crescente da s.

Undirected Single Source Shortest Paths with Positive Integer Weights in Linear Time

Mikkel Thorup
AT&T Labs—Research

The single source shortest paths problem (SSSP) is one of the classic problems in algorithmic graph theory: given a positively weighted graph G with a source vertex s, find the shortest path from s to all other vertices in the graph.

Since 1959 all theoretical developments in SSSP for general directed and undirected graphs have been based on Dijkstra's algorithm, visiting the vertices in order of increasing distance from s. Thus, any implementation of Dijkstra's algorithm sorts the vertices according to their distances from s. However, we do not know how to sort in linear time.

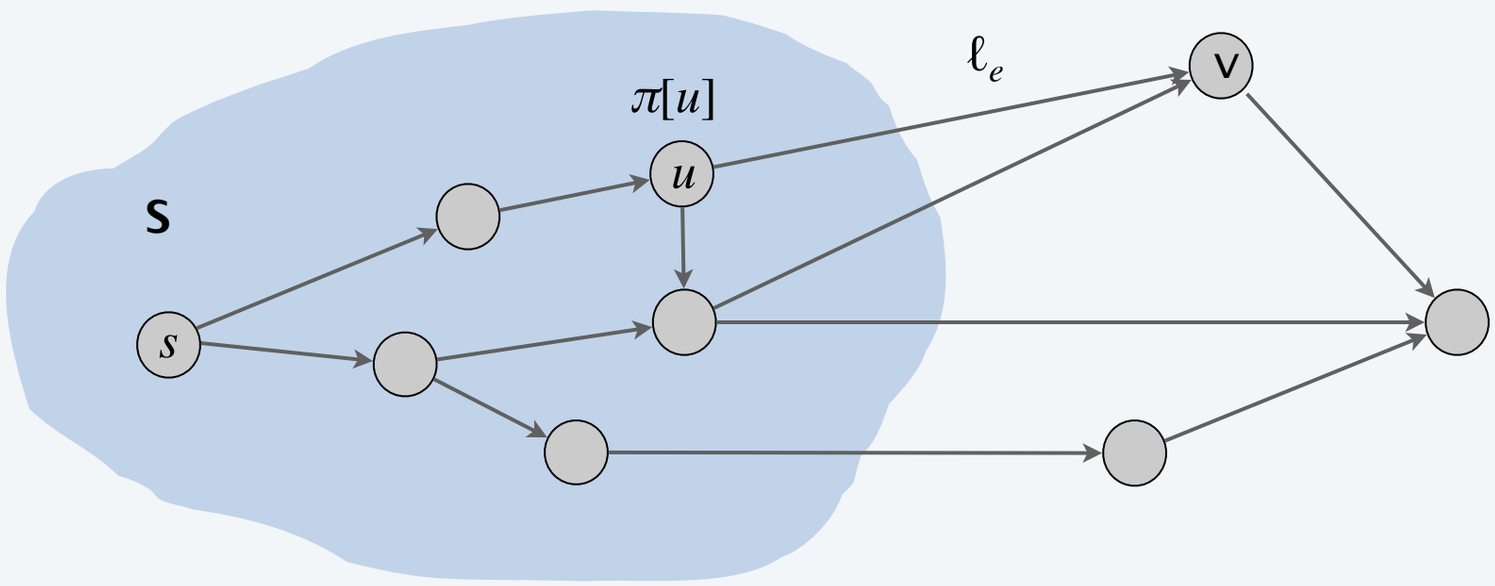
Here, a deterministic linear time and linear space algorithm is presented for the undirected single source shortest paths problem with positive integer weights. The algorithm avoids the sorting bottle-neck by building a hierarchical bucketing structure, identifying vertex pairs that may be visited in any order.

Estensioni dell'algoritmo di Dijkstra

L'algoritmo di Dijkstra e la sua analisi si estendono a vari problemi correlati:

- Cammini minimi in grafi non orientati: $\pi[v] \leq \pi[u] + \ell(u, v)$.
- Cammini a capacità massima: $\pi[v] \geq \min \{ \pi[u], c(u, v) \}$.
- Cammini ad affidabilità massima: $\pi[v] \geq \pi[u] \times \gamma(u, v)$.
- ...

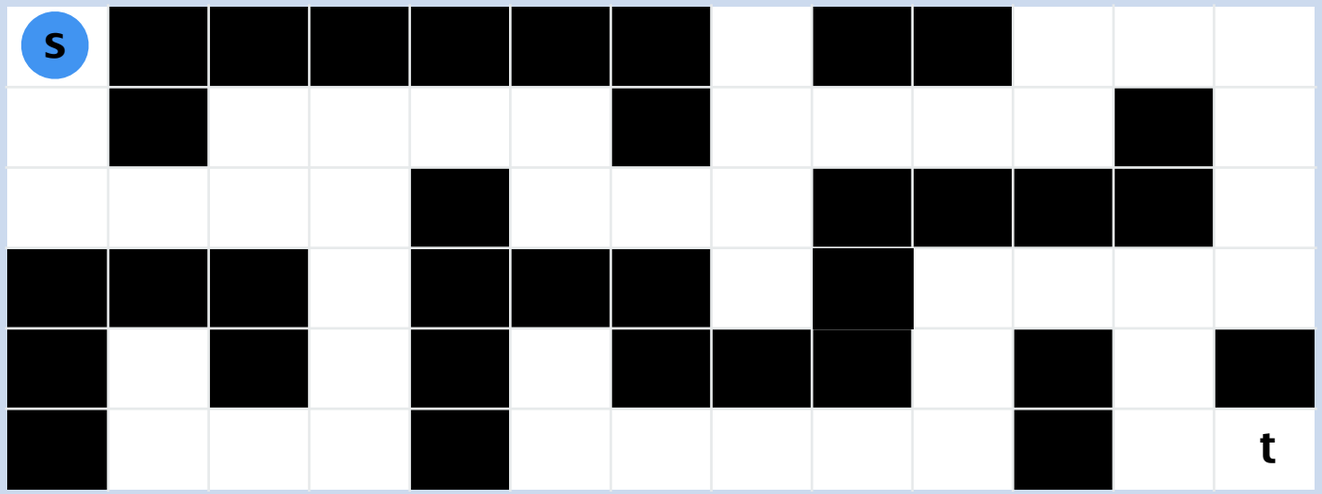
Struttura algebrica chiave. Semianello chiuso (min-plus, bottleneck, Viterbi, ...).



$$
\begin{aligned}
a + b &= b + a \\
a + (b + c) &= (a + b) + c \\
a + 0 &= a \\
a \cdot (b \cdot c) &= (a \cdot b) \cdot c \\
a \cdot 0 &= 0 \cdot a = 0 \\
a \cdot 1 &= 1 \cdot a = a \\
a \cdot (b + c) &= a \cdot b + a \cdot c \\
(a + b) \cdot c &= a \cdot c + b \cdot c \\
a^* &= 1 + a \cdot a^* = 1 + a^* \cdot a
\end{aligned}
$$

È data la mappa di una stazione spaziale, con una posizione iniziale s (cella di prigione) e una posizione finale t (capsula di salvataggio). La mappa è rappresentata come matrice di 0 e 1, dove gli 0 sono spazi attraversabili e gli 1 sono muri impenetrabili. La cella della prigione è nella posizione in alto a sinistra, $(0, 0)$ e la capsula di salvataggio è in basso a destra, $(w-1, h-1)$.

Scrivere una funzione che calcola la lunghezza di un cammino minimo dalla prigione alla capsula di salvataggio, in cui è permesso di rimuovere un muro (un singolo muro, una sola volta nel percorso).



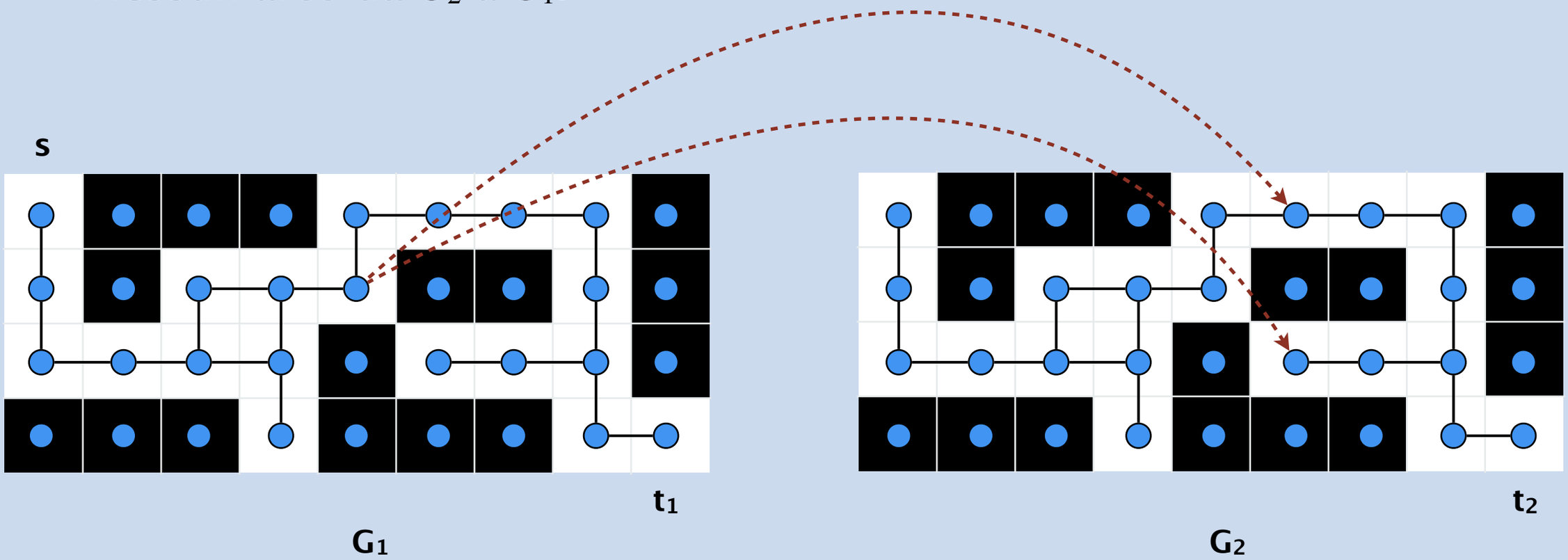
GOOGLE'S FOO.BAR CHALLENGE

Idea 1. Modelliamo il labirinto come sottografo di un grafo "griglia".

- Un nodo per ogni cella.
- Un arco simmetrico tra due celle aperte adiacenti (lunghezza = 1).

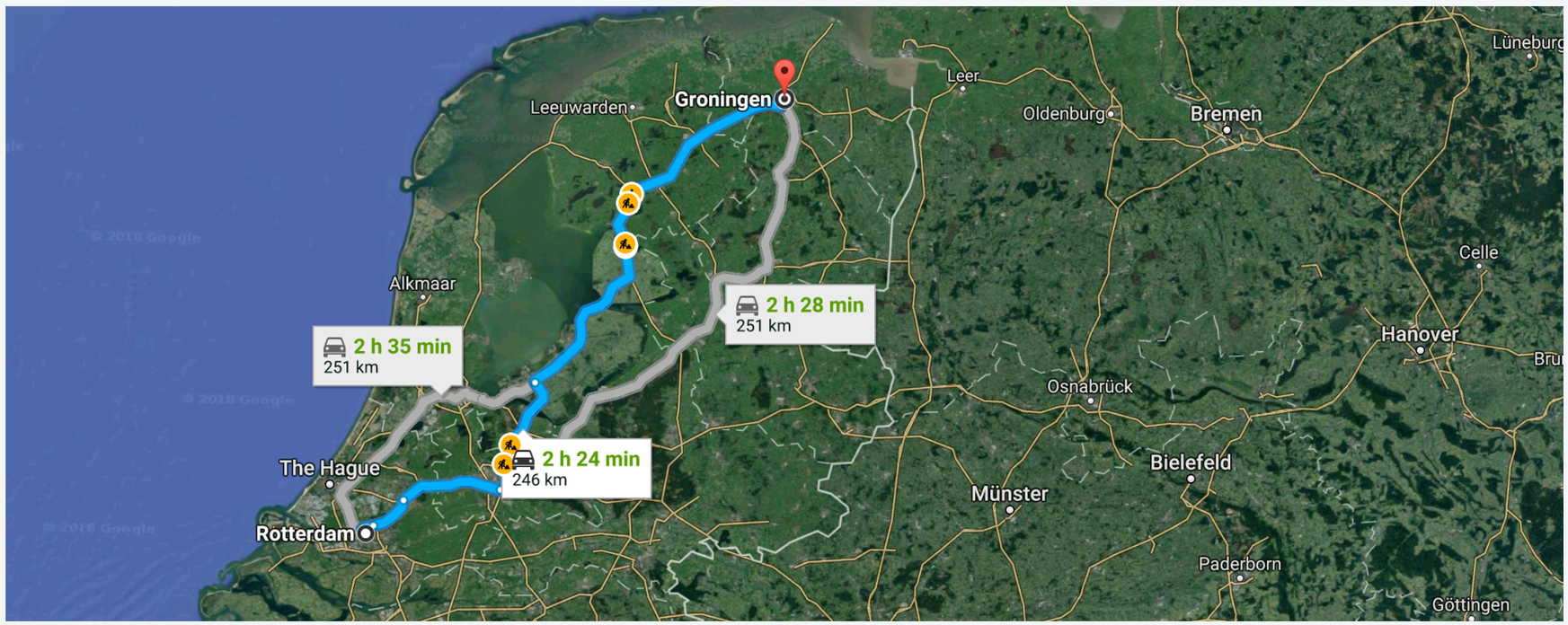
Idea 2. Usiamo un trucco in cui "duplichiamo" il grafo.

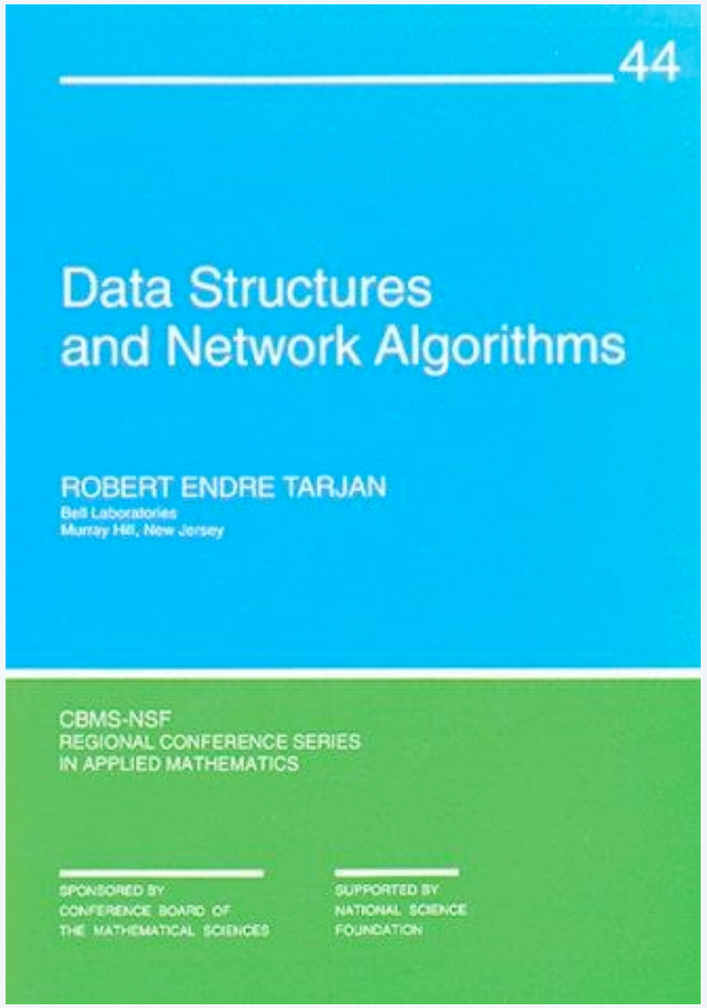
- Un arco da G_1 a G_2 corrisponde a rimuovere un muro (lunghezza = 2).
- Nessun arco da G_2 a G_1.



Edsger Dijkstra

> *“ Qual è la via più breve per andare da Rotterdam a Groningen? È data dall'algoritmo del cammino minimo, che ho inventato in circa 20 minuti. Una mattina facevo shopping ad Amsterdam con la mia fidanzata, e, stanchi, ci siamo seduti sulla terrazza di un bistrò a bere una tazza di caffè mentre pensavo proprio a come far questo. È allora che ho inventato l'algoritmo per il cammino minimo.”* — Edsger Dijsktra





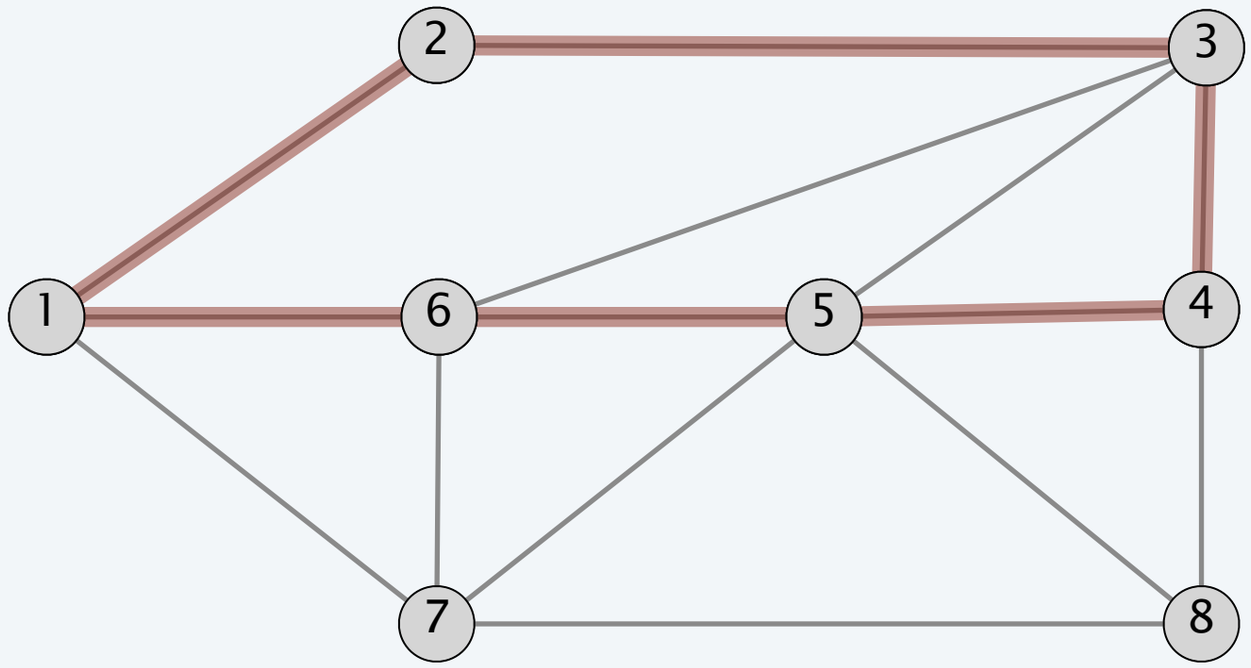
SECTION 6.1

4. ALGORITMI AVIDI II

- *Dijkstra's algorithm*
- *alberi ricoprenti minimi*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Cicli

Def. Un ciclo è un cammino della forma *a*-*b*, *b*-*c*, *c*-*d*, …, *y*-*z*, *z*-*a*.



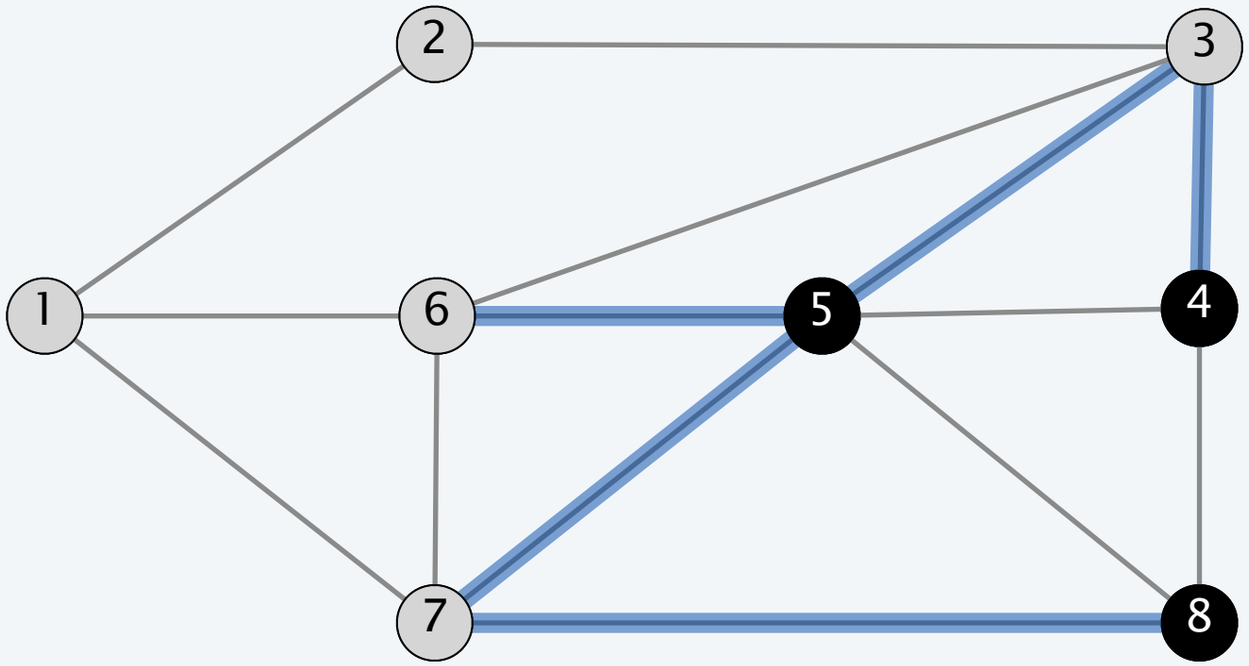
cammino P = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6) }

ciclo C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

Tagli [cut]

Def. Un taglio [cut] è una partizione dei nodi in due sottoinsiemi non vuoti S e $V - S$.

Def. Il cutset di un taglio S è l'insieme degli archi con esattamente un estremo in S.



taglio S = { 4, 5, 8 }

cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

Alberi ricoprenti minimi: quiz 1

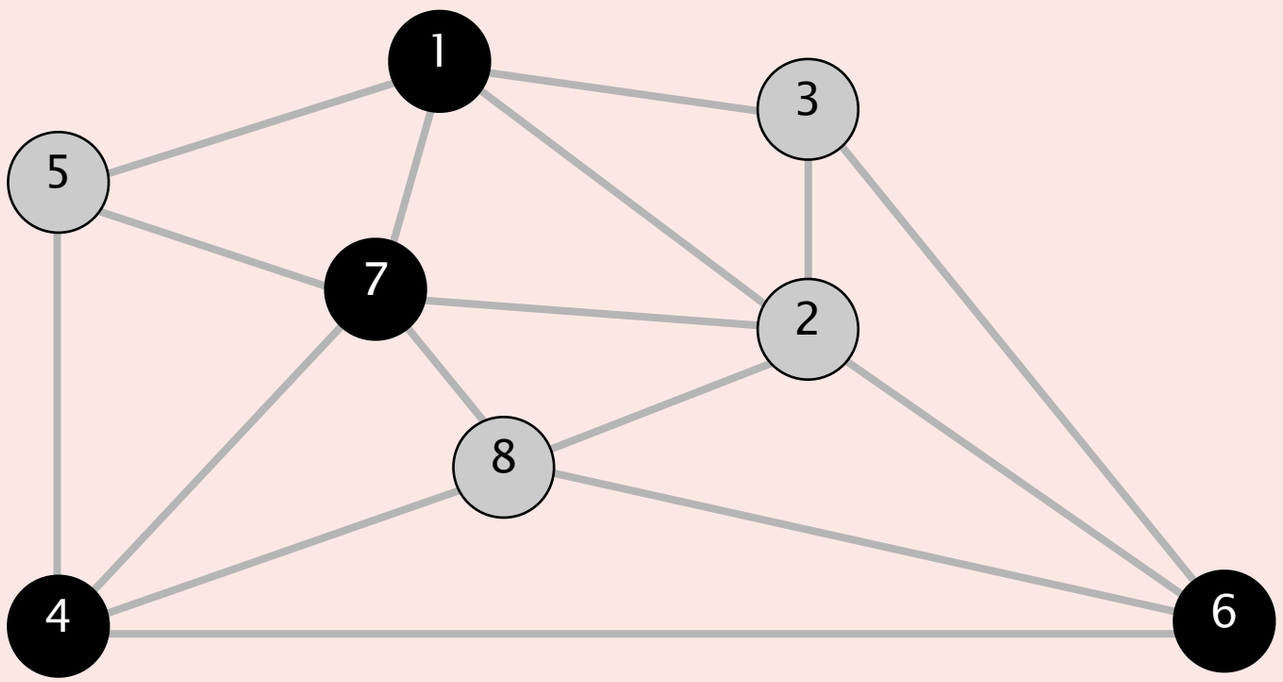
Consideriamo il taglio S = { 1, 4, 6, 7 }. Quale arco è nel cutset di S?

A. S non è un taglio (non è connesso)

B. 1–7

C. 5–7

D. 2–3



Alberi ricoprenti minimi: quiz 2

Sia C un ciclo e sia D un cutset. Quanti archi hanno in comune C e D? Scegliere la risposta migliore.

A. 0

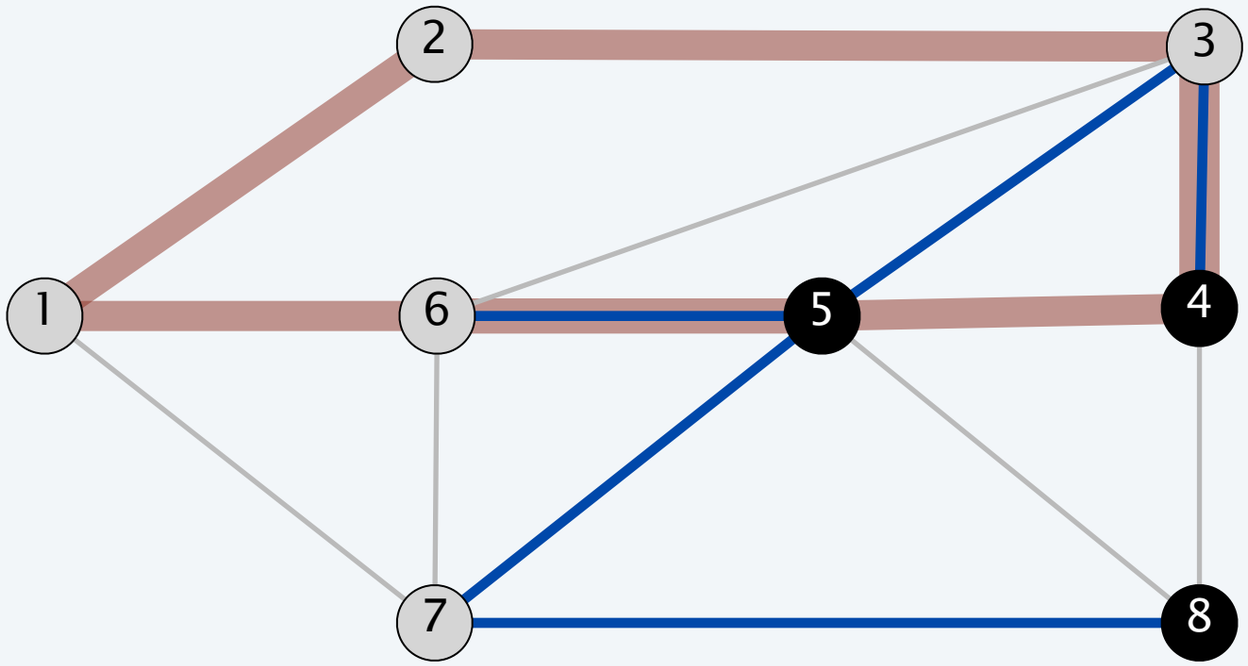
B. 2

C. non 1

D. un numero pari

Intersezione ciclo-taglio

Proposizione. Un ciclo ed un cutset si intersecano in un numero pari di archi.



ciclo C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

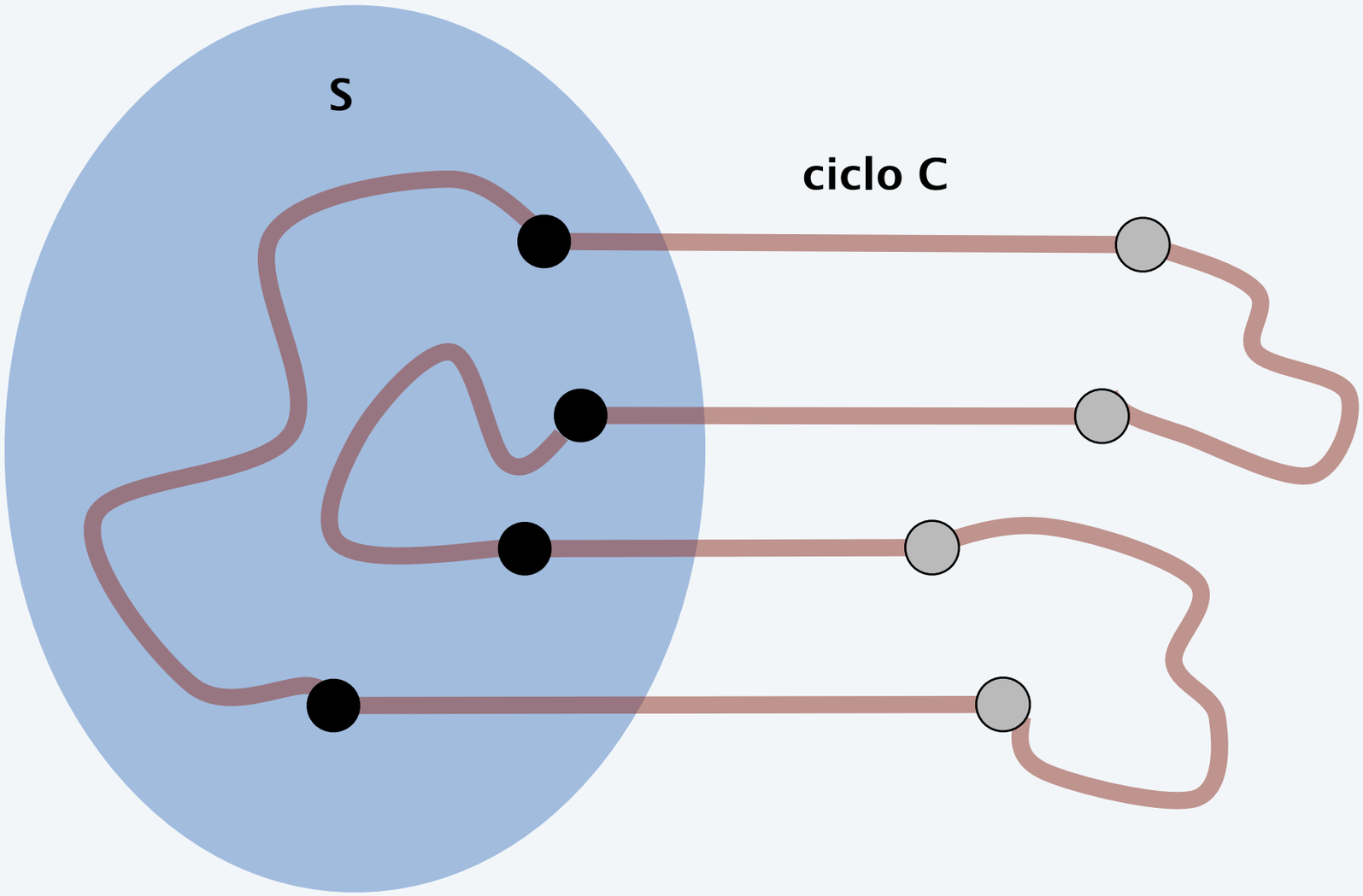
cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

intersezione C ∩ D = { (3, 4), (5, 6) }

Intersezione ciclo-taglio

Proposizione. Un ciclo ed un cutset si intersecano in un numero pari di archi.

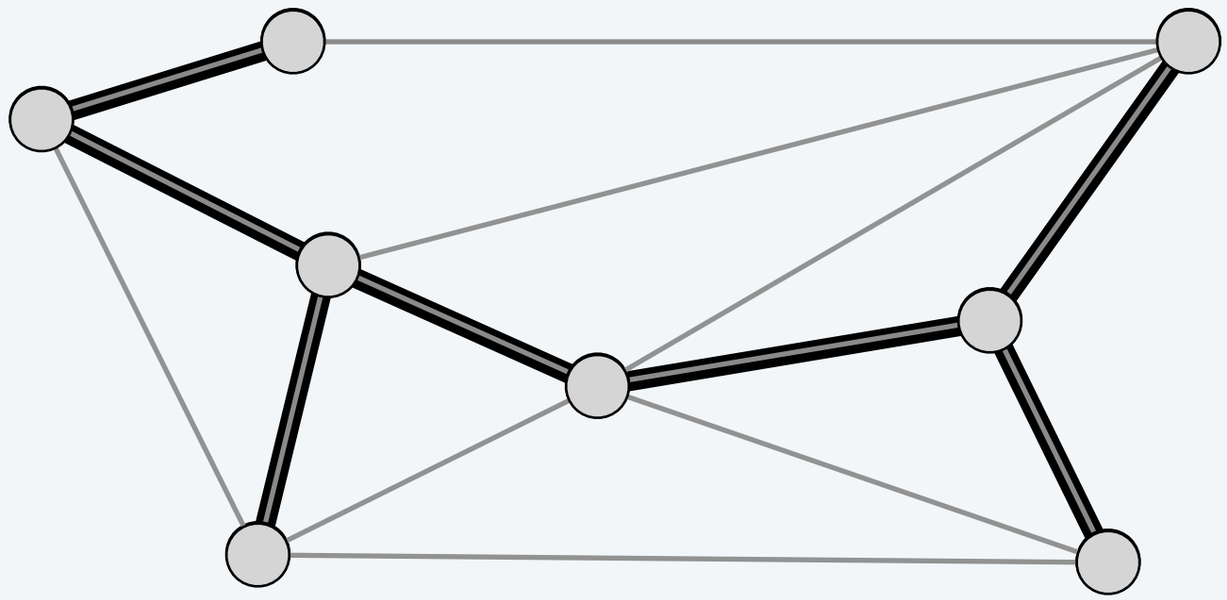
Dim. [via figura]



Definizione di albero ricoprente

Def. Sia $H = (V, T)$ un sottografo di un grafo non orientato $G = (V, E)$.

H è un albero ricoprente di G se H è sia aciclico che connesso.



grafo G = (V, E)

albero ricoprente H = (V, T)

Alberi ricoprenti minimi: quiz 3

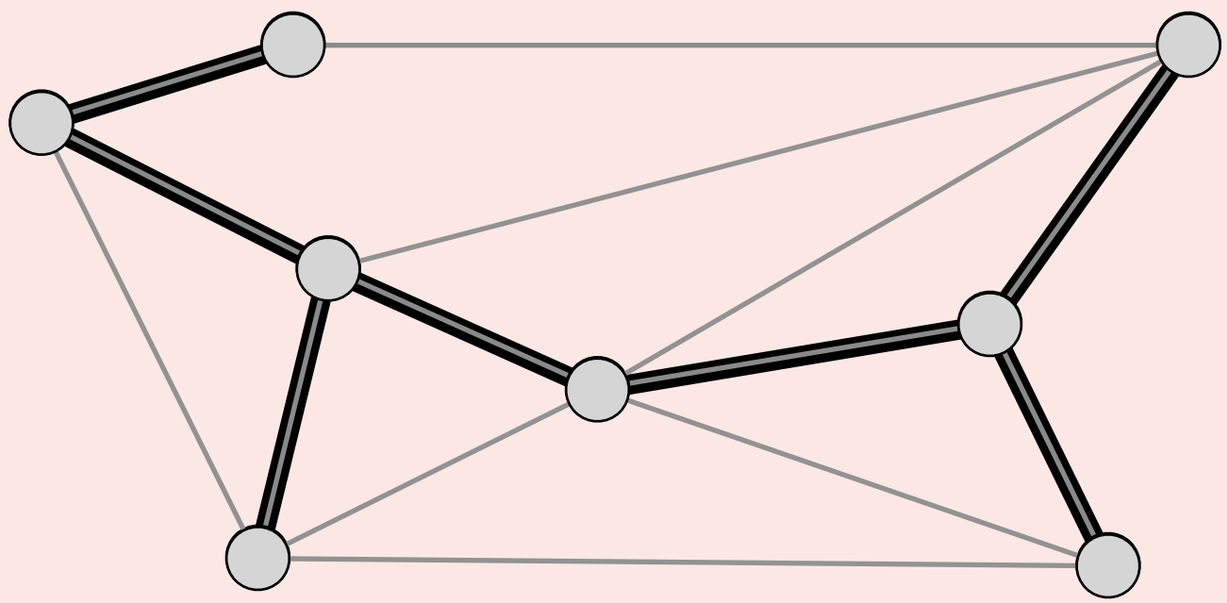
Quale delle seguenti proprietà è vera per ogni albero ricoprente H?

A. Contiene esattamente $|V| - 1$ archi.

B. La rimozione di un arco qualunque lo disconnette.

C. L'aggiunta di un arco qualunque crea un ciclo.

D. Tutte le precedenti.



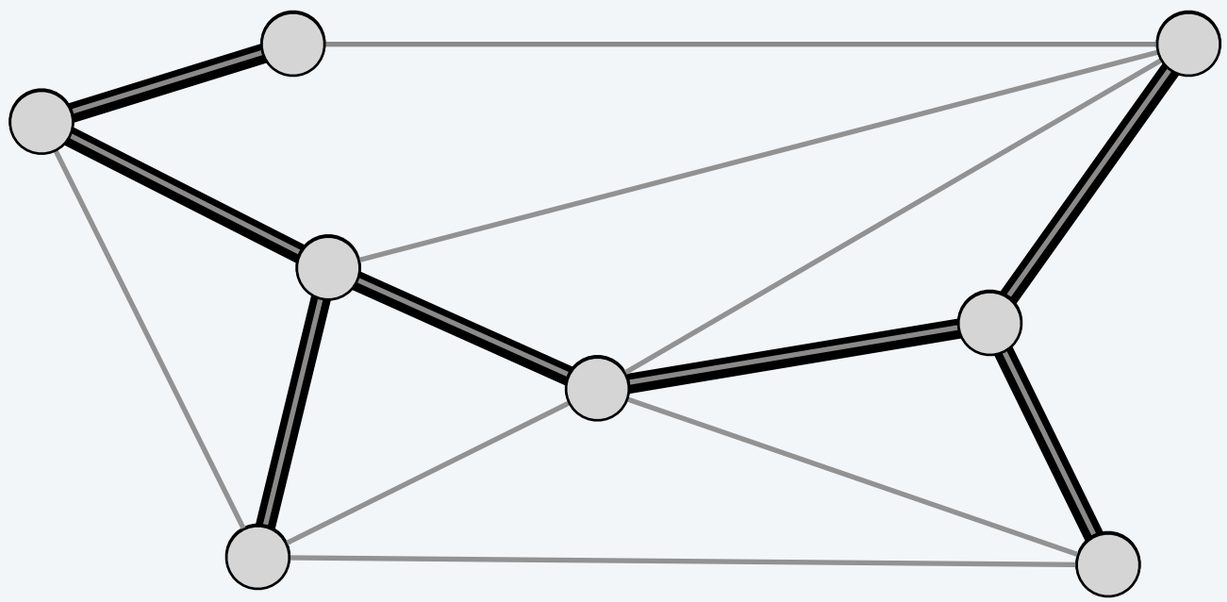
grafo G = (V, E)

albero ricoprente H = (V, T)

Proprietà degli alberi ricoprenti

Proposizione. Sia $H = (V, T)$ un sottografo di un grafo $G = (V, E)$. Le seguenti sono equivalenti:

- H è un albero ricoprente di G.
- H è aciclico e connesso.
- H è connesso ed ha $|V| - 1$ archi.
- H è aciclico ed ha $|V| - 1$ archi.
- H è minimamente connesso: cancellare qualunque arco lo disconnette.
- H è massimamente aciclico: aggiungere qualunque arco crea un ciclo.

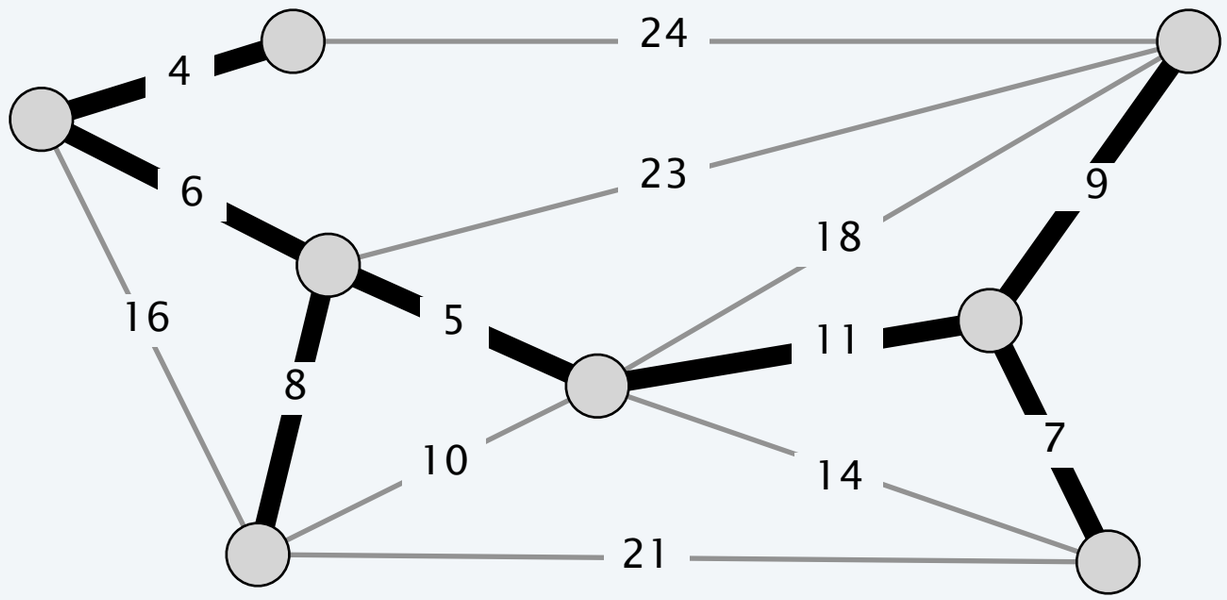


grafo G = (V, E)

albero ricoprente H = (V, T)

Albero ricoprente minimo (MST – Minimum Spanning Tree)

Def. Dato un grafo non orientato connesso $G = (V, E)$ con costi sugli archi c_e, un albero ricoprente minimo (V, T) è un albero ricoprente di G che minimizza la somma dei costi degli archi in T.



costo MST = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

Teorema di Cayley. Il grafo completo su n nodi ha n^{n-2} alberi ricoprenti.

non possiamo sperare di trovarlo per enumerazione

Alberi ricoprenti minimi: quiz 4

Supponiamo di cambiare il costo di ogni arco di G come segue. Quando si può dire che ogni MST di G è un MST di G' (e viceversa)? Si assuma c(e) > 0 per ogni e.

A. $c'(e) = c(e) + 17$.

B. $c'(e) = 17 \times c(e)$.

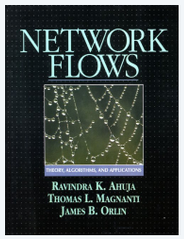
C. $c'(e) = \log_{17} c(e)$.

D. Tutte le precedenti.

Applicazioni

MST è un problema fondamentale con svariate applicazioni.

- Elaborazione di immagini (Dithering).
- Analisi di cluster.
- Cammini a collo di bottiglia massimo.
- Verifica facciale in tempo reale.
- Codici LDPC per la correzione d'errore.
- Identificazione di reti stradali in fotografie satellitari ed aeree.
- Riduzione della memoria usata nel sequenziamento di aminoacidi in proteina.
- Protocolli Ethernet che evitano cicli nell'infrastruttura di rete.
- Algoritmi approssimati per problemi NP-ardui (e.g., TSP, Steiner tree).
- Progetto di reti (di comunicazioni, elettriche, idrauliche, informatiche, stradali).

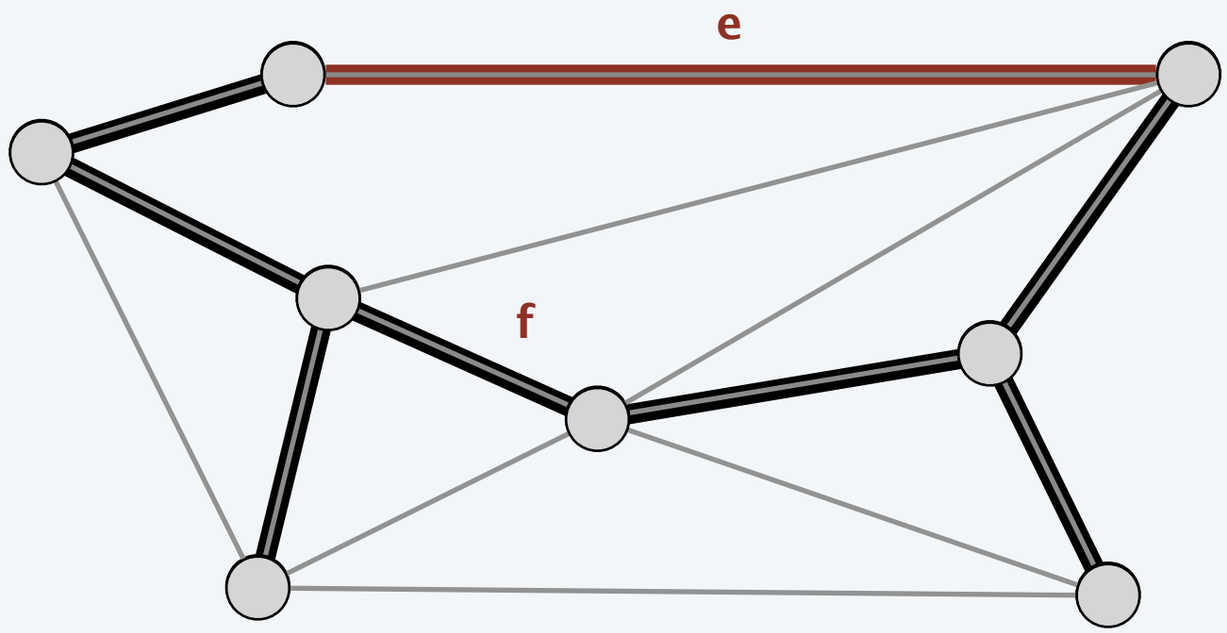


Network Flows: Theory, Algorithms, and Applications, by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

Ciclo fondamentale

Ciclo fondamentale. Sia $H = (V, T)$ un albero ricoprente di $G = (V, E)$.

- Per ogni arco $e \in E$ non nell'albero: $T \cup \{ e \}$ contiene un solo ciclo, sia esso C.
- Per ogni arco $f \in C$: $(V, T \cup \{ e \} - \{ f \})$ è un albero ricoprente.



grafo G = (V, E)

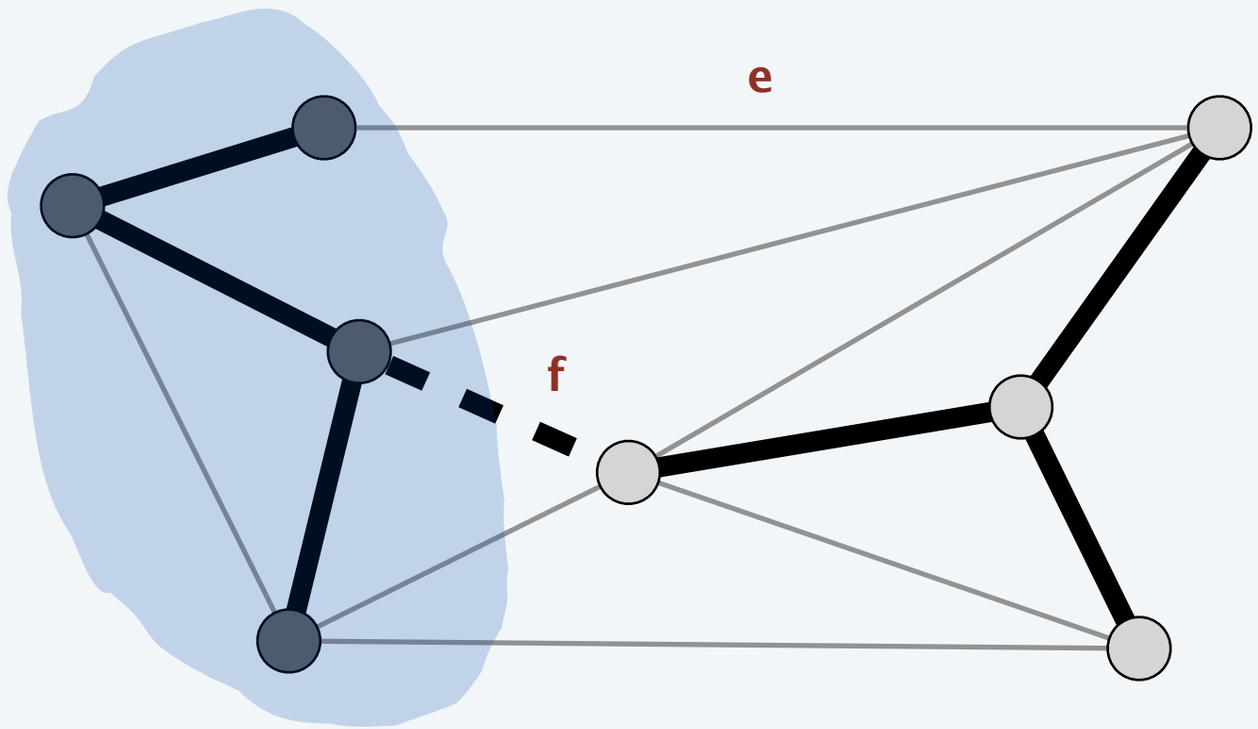
albero ricoprente H = (V, T)

Osservazione. Se $c_e < c_f$, allora (V, T) non è un MST.

Cutset fondamentale

Cutset fondamentale. Sia $H = (V, T)$ un albero ricoprente di $G = (V, E)$.

- Per ogni arco $f \in T$: $(V,\ T - \{ f \})$ ha due componenti connesse.
- Sia D il cutset corrispondente.
- Per ogni arco $e \in D$: $(V,\ T - \{ f \} \cup \{ e \})$ è un albero ricoprente.



grafo G = (V, E)

albero ricoprente H = (V, T)

Osservazione. Se $c_e < c_f$, allora (V, T) non è un MST.

L'algoritmo avido

Regola rossa.

- Sia C un ciclo senza archi rossi.
- Scegli un arco non colorato di C a costo massimo e coloralo di rosso.

Regola blu.

- Sia D un cutset senza archi blu.
- Scegli un arco non colorato di D a costo minimo e coloralo di blu.

Algoritmo avido.

- Applica le regole rossa e blu (in qualunque ordine!) finché non sono stati colorati tutti gli archi. Gli archi blu formano un MST.
- Nota: può essere terminato dopo che $n - 1$ archi sono stati colorati blu.

Algoritmo avido: dimostrazione di correttezza

Invariante di colore. Esiste un MST (V, T^*) contenente ogni arco blu e nessun arco rosso.

Dim. [per induzione sul numero di iterazioni]

Caso base. Nessun arco è colorato $\Rightarrow$ qualunque MST soddisfa l'invariante.

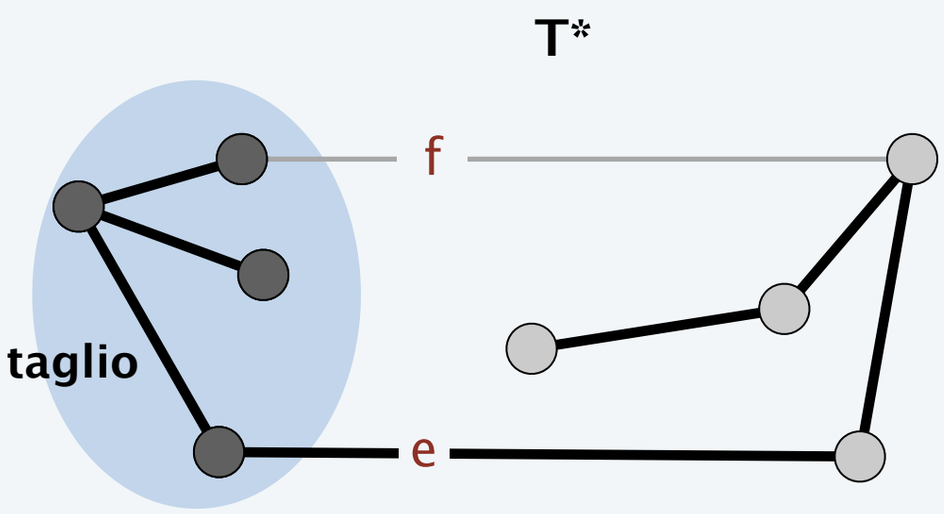
Algoritmo avido: dimostrazione di correttezza

Invariante di colore. Esiste un MST (V, T^*) contenente ogni arco blu e nessun arco rosso.

Dim. [per induzione sul numero di iterazioni]

Passo induttivo (regola blu). Supponiamo l'invariante sia valido prima di applicare la regola blu.

- Sia D il cutset scelto, e sia f l'arco colorato in blu.
- Se $f \in T^*$, allora T^* soddisfa ancora l'invariante.
- Altrimenti, consideriamo il ciclo C ottenuto aggiungendo f a T^*.
- Sia $e \in C$ un altro arco in D.
- e non è colorato e $c_e \geq c_f$, poiché:
  - $e \in T^* \Rightarrow e$ non è rosso
  - regola blu $\Rightarrow e$ non è blu e $c_e \geq c_f$
- Quindi, $T^* \cup \{f\} - \{e\}$ soddisfa l'invariante.



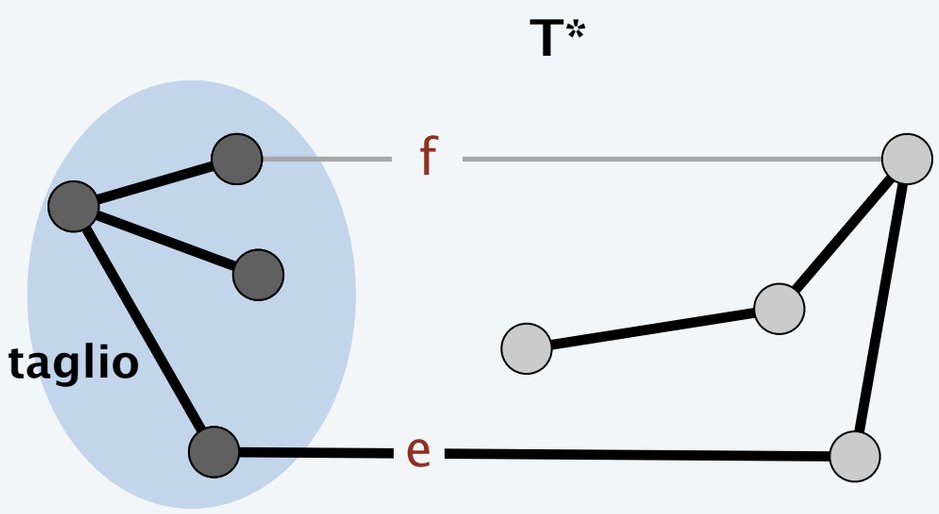
Algoritmo avido: dimostrazione di correttezza

Invariante di colore. Esiste un MST (V, T^*) contenente ogni arco blu e nessun arco rosso.

Dim. [per induzione sul numero di iterazioni]

Passo induttivo (regola rossa). Supponiamo l'invariante sia valido prima di applicare la regola rossa.

- Sia C il ciclo scelto, e sia e l'arco colorato in rosso.
- Se $e \notin T^*$, allora T^* soddisfa ancora l'invariante.
- Altrimenti, consideriamo il cutset D ottenuto rimuovendo e da T^*.
- Sia $f \in D$ un altro arco in C.
- f non è colorato e $c_e \geq c_f$, poiché:
  - $f \notin T^* \Rightarrow f$ non è blu
  - regola rossa $\Rightarrow f$ non è rosso e $c_e \geq c_f$
- Quindi, $T^* \cup \{f\} - \{e\}$ soddisfa l'invariante. ▪



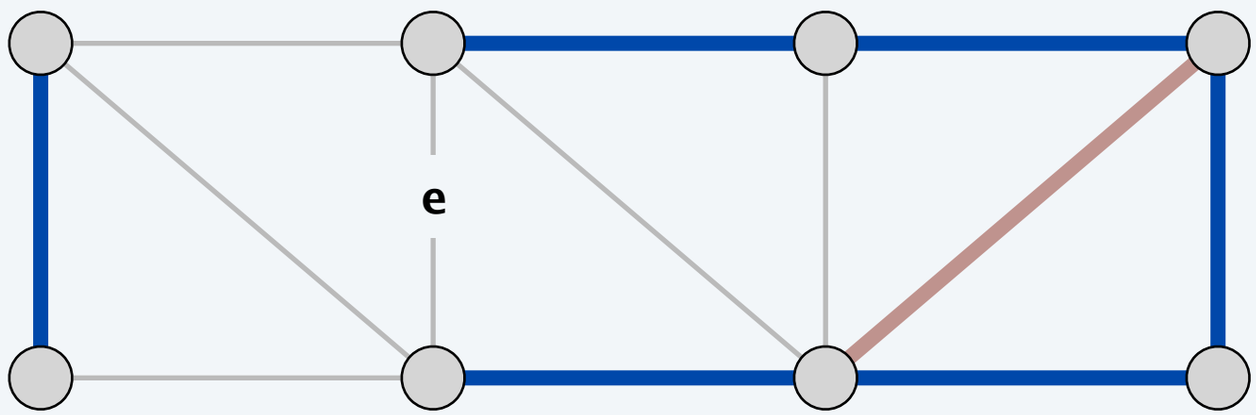
Algoritmo avido: dimostrazione di correttezza

Teorema. L'algoritmo avido termina. Gli archi blu formano un MST.

Dim. Dobbiamo mostrare che si può sempre applicare almeno una delle due regole di colorazione.

- Supponiamo l'arco e non sia stato colorato.
- Gli archi blu formano una foresta (grafo aciclico).
- Caso 1: entrambi gli estremi di e sono nello stesso albero blu.

  $\Rightarrow$ si può applicare la regola rossa al ciclo formato aggiungendo e alla foresta blu.



Caso 1

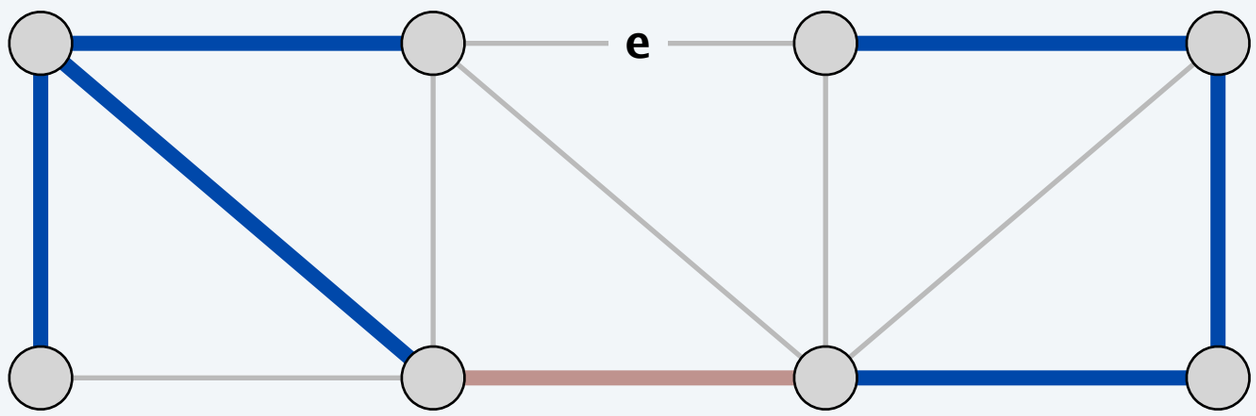
Algoritmo avido: dimostrazione di correttezza

Teorema. L'algoritmo avido termina. Gli archi blu formano un MST.

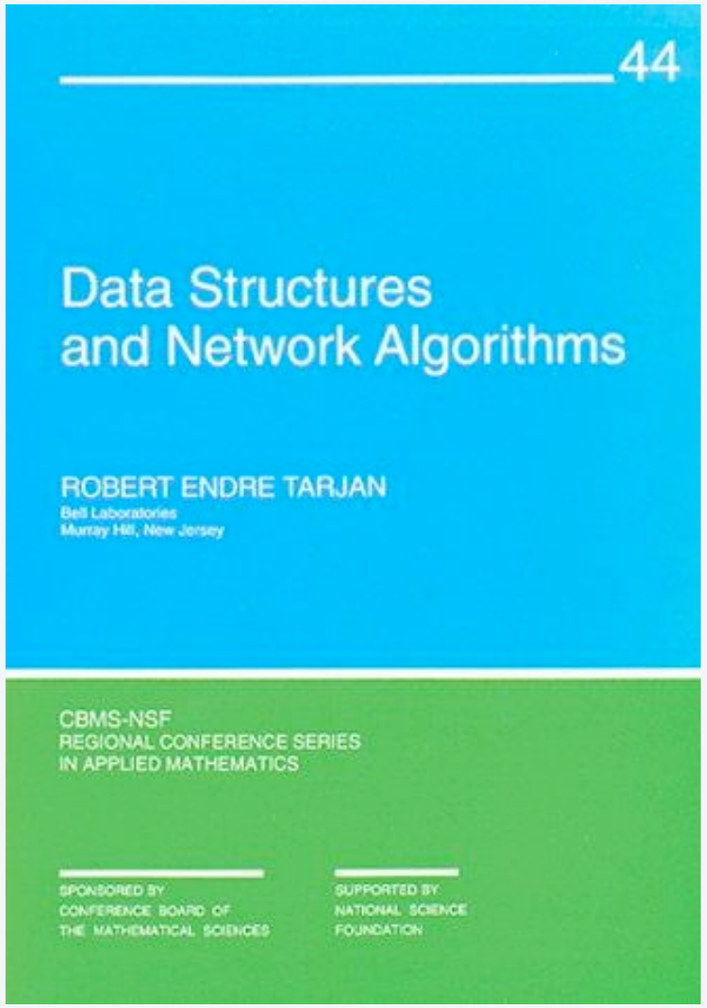
Dim. Dobbiamo mostrare che si può sempre applicare almeno una delle due regole di colorazione.

- Supponiamo l'arco e non sia stato colorato.
- Gli archi blu formano una foresta (grafo aciclico).
- Caso 2: gli estremi di e sono in alberi blu distinti.

  $\Rightarrow$ si può applicare la regola blu al cutset indotto da uno dei due alberi blu. ▪



Caso 2



SECTION 6.2

4. ALGORITMI AVIDI II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *single-link clustering*
- *min-cost arborescences*

Algoritmo di Prim [algoritmo di Jarnik]

Inizializza $S = \{ s \}$ per un qualche nodo s, $T = \emptyset$.

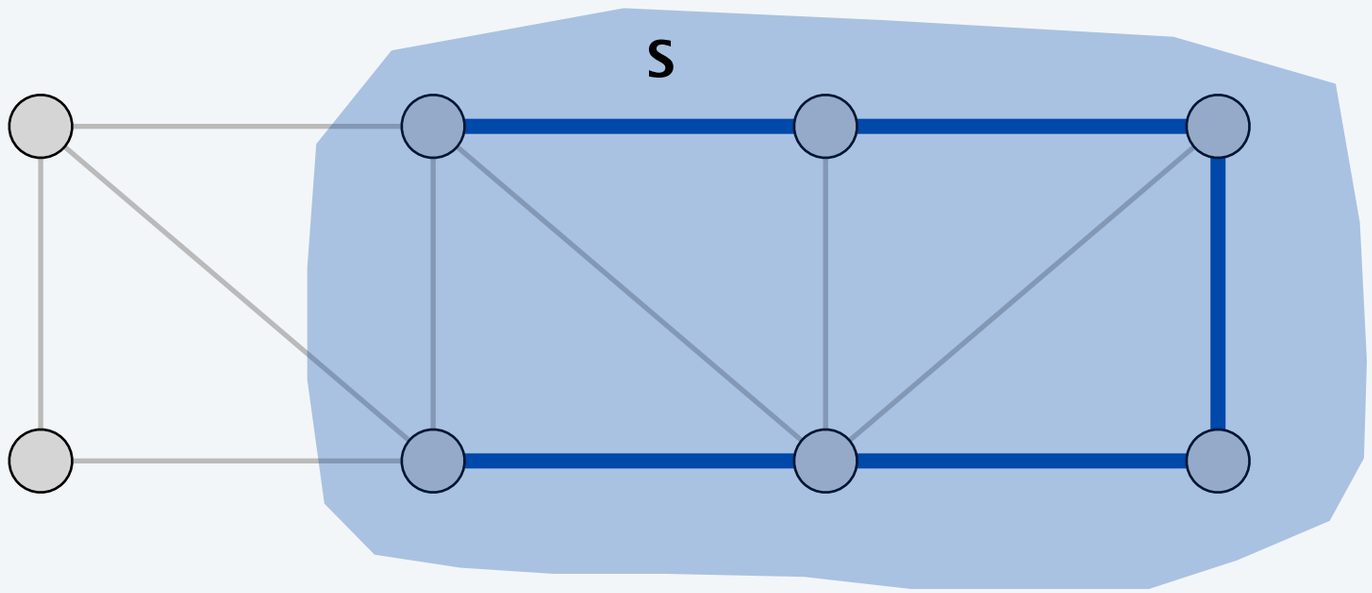
Ripeti $n - 1$ volte:

- Aggiungi a T un arco a costo minimo con esattamente un estremo in S.
- Aggiungi l'altro estremo ad S.

per costruzione, gli archi del cutset non sono mai colorati

Teorema. L'algoritmo di Prim calcola un MST.

Dim. Caso particolare dell'algoritmo di colorazione (usa ripetutamente la regola blu su S). ▪



Algoritmo di Prim: implementazione

Teorema. L'algoritmo di Prim può essere implementato in tempo $O(m \log n)$.

Dim. L'implementazione è quasi identica all'algoritmo di Dijkstra.

PRIM (V, E, c)

$S \leftarrow \emptyset$, $T \leftarrow \emptyset$.

$s \leftarrow$ un qualunque nodo di V.

FOREACH $v \neq s$: $\pi[v] \leftarrow \infty, pred[v] \leftarrow null$; $\pi[s] \leftarrow 0$.

Crea una coda di priorità vuota pq.

FOREACH $v \in V$: INSERT$(pq, v, \pi[v])$.

WHILE (IS-NOT-EMPTY(pq))

  $u \leftarrow$ DEL-MIN(pq).

  $S \leftarrow S \cup \{ u \}$, $T \leftarrow T \cup \{ pred[u] \}$.

  FOREACH arco $e = (u, v) \in E$ con $v \notin S$:

    IF $(c_e < \pi[v])$

      DECREASE-KEY(pq, v, c_e).

      $\pi[v] \leftarrow c_e$; $pred[v] \leftarrow e$.

$\pi[v]$ = costo del più economico arco noto tra v ed S

Algoritmo di Kruskal

Considera gli archi in ordine ascendente di costo:

- Aggiungili all'albero a meno che ciò non crei un ciclo.

Teorema. L'algoritmo di Kruskal calcola un MST.

Dim. Caso particolare dell'algoritmo di colorazione.

- Caso 1: entrambi gli estremi di e sono nello stesso albero blu.

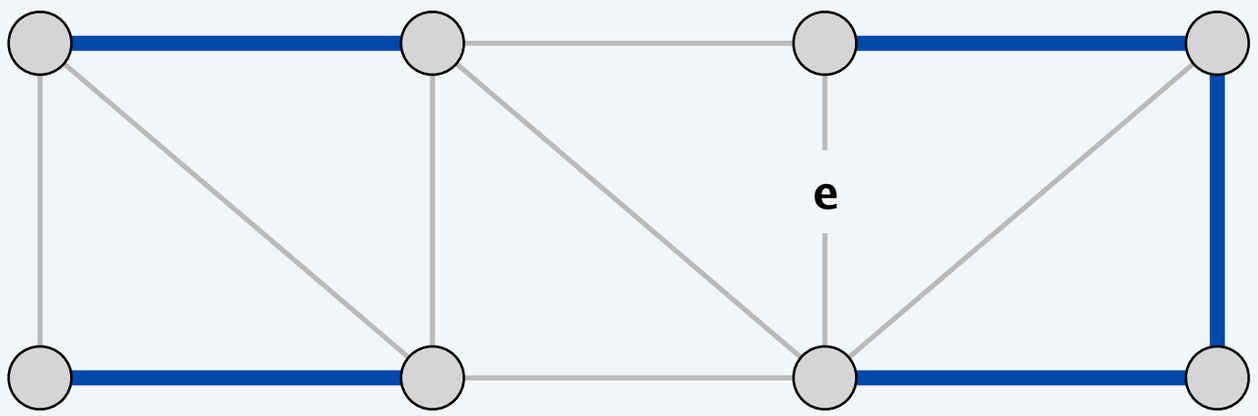
  $\Rightarrow$ colora e rosso applicando la regola rossa al ciclo corrispondente.

tutti gli altri archi del ciclo sono blu

- Caso 2: gli estremi di e sono in alberi blu distinti.

  $\Rightarrow$ colora e blu applicando la regola blu al cutset definito da uno dei due alberi. ▪

nessun arco del cutset ha costo inferiore
(poiché Kruskal lo ha scelto per primo)



Struttura dati Union-Find

Union-Find. Una struttura dati dinamica che mantiene sottoinsiemi disgiunti di un universo di n elementi.

operazione	descrizione	tempo di esecuzione per un universo di n elementi (implementazione basata su array)
MAKE-SET(v)	crea un insieme contenente solo v	O(1)
CONNECTED(u, v)	gli elementi u e v sono nello stesso insieme?	O(log n)
UNION(u, v)	fondi i due insiemi contenenti u e v	O(log n) †

† costo ammortizzato su n operazioni

Algoritmo di Kruskal: implementazione

Teorema. L'algoritmo di Kruskal può essere implementato per eseguire in tempo $O(m \log n)$.

- Ordiniamo gli archi per costo.
- Usiamo una struttura dati union–find per mantenere dinamicamente le componenti connesse

KRUSKAL (V, E, c)

ORDINA m archi per costo in modo che $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$.

$T \leftarrow \emptyset$.

FOREACH $v \in V$: MAKE-SET(v).

FOR $i = 1$ TO m

  $(u, v) \leftarrow e_i$.

  IF (NOT CONNECTED(u, v)) ← u e v sono nella stessa componente o no?

    $T \leftarrow T \cup \{ e_i \}$.

    UNION(u, v). ← unisci u e v in una stessa componente

RETURN T.

Algoritmo di cancellazione a rovescio

Inizia con tutti gli archi in T e considerali in ordine decrescente di costo:

- Cancella ciascun arco da T a meno che ciò non disconnetta T.

Teorema. L'algoritmo di cancellazione a rovescio calcola un MST.

Dim. Caso particolare dell'algoritmo di colorazione.

- Caso 1. [cancellare e non sconnette T]

  $\Rightarrow$ applica la regola rossa al ciclo C formato aggiungendo e ad un altro cammino in T tra i suoi due estremi

nessun arco di C è più costoso
(sarebbe stato considerato e cancellato in precedenza)

- Caso 2. [cancellare e sconnette T]

  $\Rightarrow$ applica la regola blu al cutset D indotto da una delle componenti ▪

e è l'unico arco restante del cutset
(tutti gli altri archi di D devono esser stati colorati in rosso o cancellati)

Fatto. [Thorup 2000] Può essere implementato per eseguire in tempo $O(m \log n (\log \log n)^3)$ time.

Riepilogo: algoritmo avido per il MST

Regola rossa.

- Sia C un ciclo senza archi rossi.
- Scegli un arco non colorato di C a costo massimo e coloralo di rosso.

Regola blu.

- Sia D un cutset senza archi blu.
- Scegli un arco non colorato di D a costo minimo e coloralo di blu.

Algoritmo avido.

- Applica le regole rossa e blu (in qualunque ordine!) finché non sono stati colorati tutti gli archi. Gli archi blu formano un MST.
- Nota: può essere terminato dopo che $n - 1$ archi sono stati colorati blu.

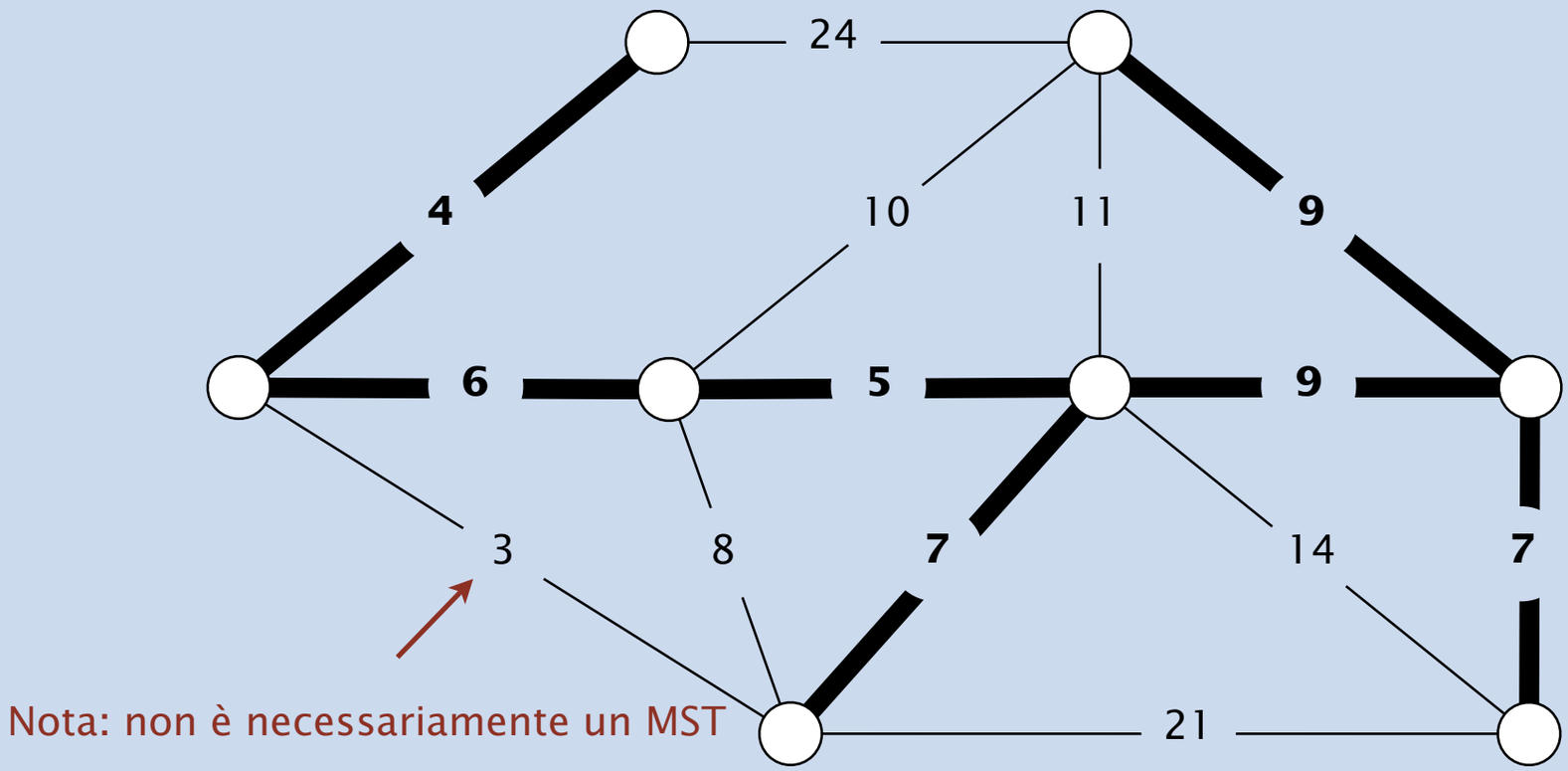
Teorema. L'algoritmo avido è corretto.

Casi particolari. Prim, Kruskal, cancellazione a rovescio, ...

Albero Ricoprente a Ingorgo Minimo [MBST]

Problema. Dato un grafo connesso G con archi di costo positivo, trovare un albero ricoprente che minimizza il costo dell'arco più costoso.

Obiettivo. Tempo $O(m \log n)$ o migliore.



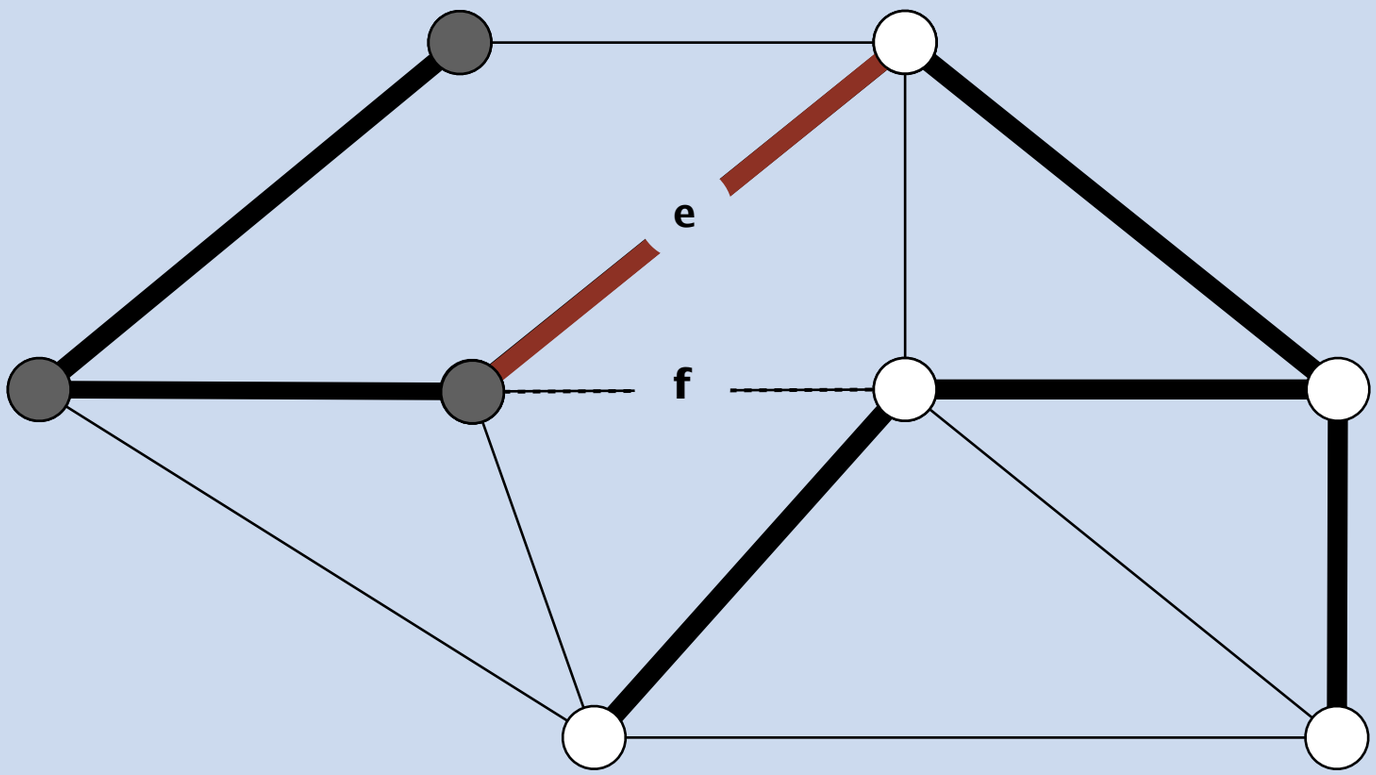
albero ricoprente a ingorgo minimo T (valore = 9)

Albero Ricoprente a Ingorgo Minimo [MBST]

Soluzione 1. Calcola un MST T^*. È anche un MBST.

Dim. Supponiamo per assurdo che T^* non sia un MBST.

- Sia $e \in T^*$ un arco con costo maggiore del valore di un MBST.
- Consideriamo il taglio ottenuto rimuovendo e da T^*.
- MBST contiene almeno un arco f nel cutset.
- $T^* \cup \{ f \} - \{ e \}$ fornisce un miglior MST. ※



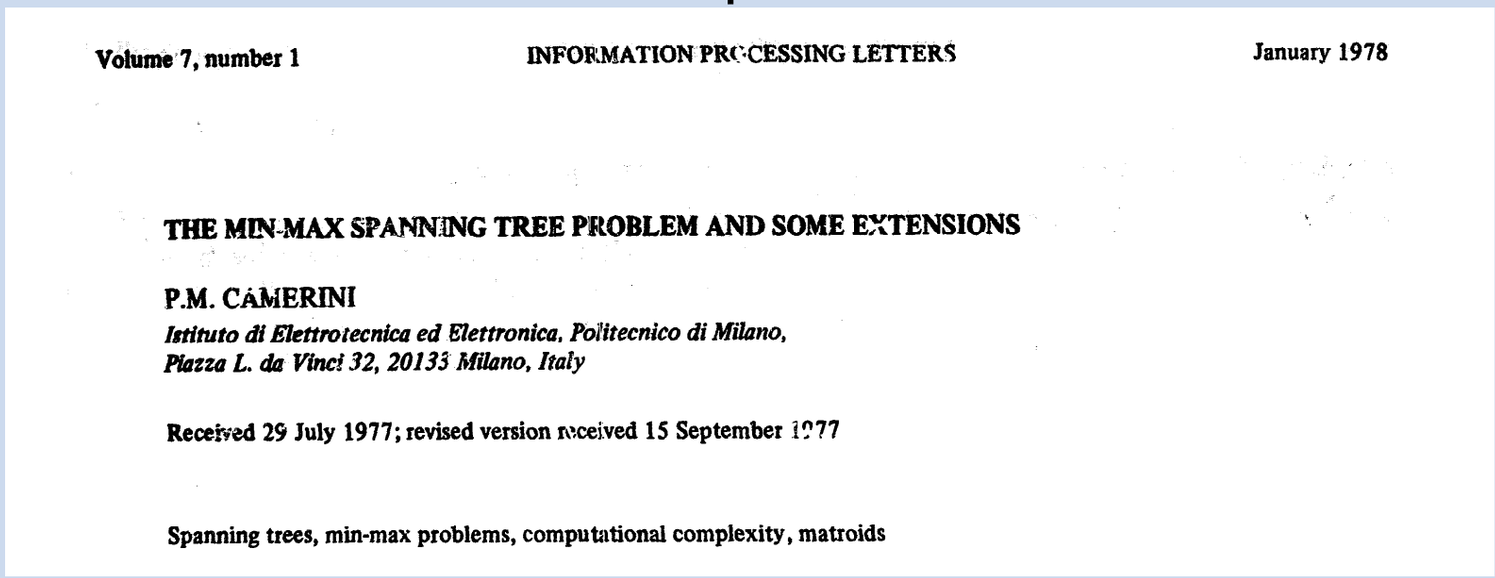
albero ricoprente minimo T*

Albero Ricoprente a Ingorgo Minimo [MBST]

Soluzione 2.

- Ordiniamo gli archi per costo crescente, $e_1 \leq e_2 \leq \ \ldots \leq e_m$.
- Sia $E_x = \ \{\, e \in E : c_e \leq x \,\}$ e $G_x = \ (V, E_x)$.
- Usiamo la ricerca binaria per trovare il minimo valore di x per il quale G_x è connesso.

Nota. L'algoritmo può essere reso tempo $O(m)$ usando algoritmi tempo-lineari per il calcolo della mediana e per le contrazioni di archi.



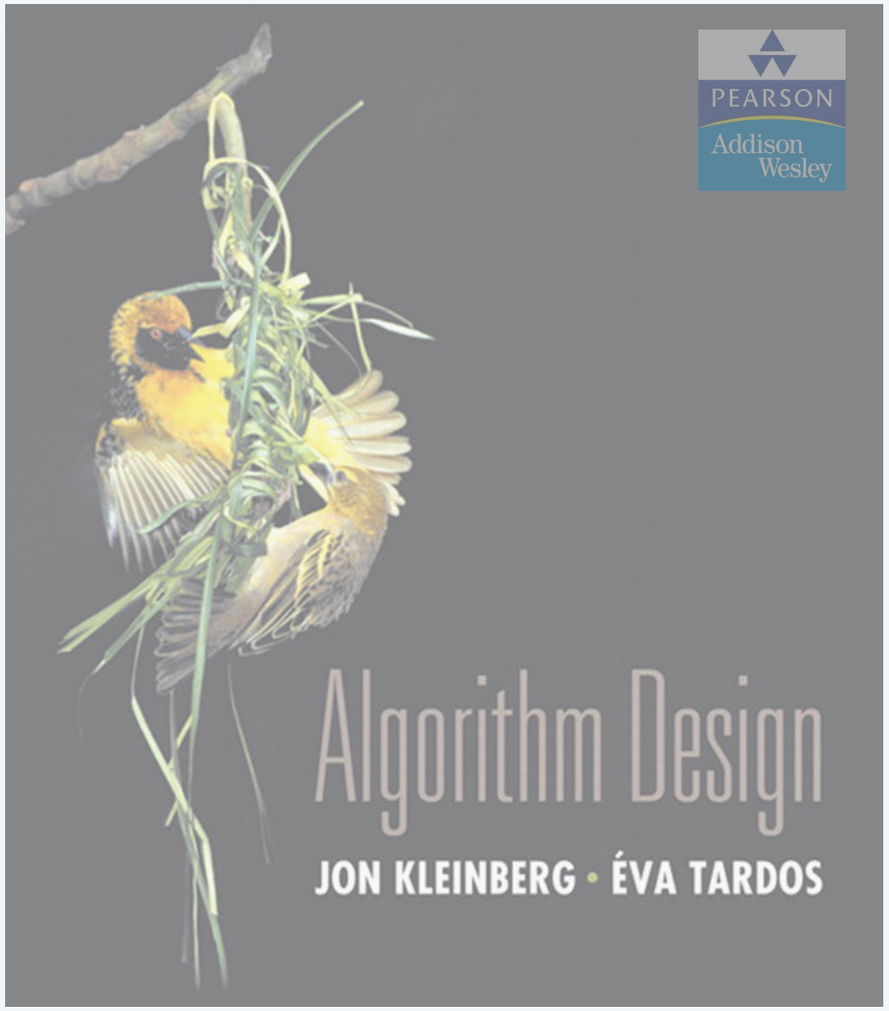
Volume 7, number 1 INFORMATION PROCESSING LETTERS January 1978

THE MIN-MAX SPANNING TREE PROBLEM AND SOME EXTENSIONS

P.M. CAMERINI
Istituto di Elettrotecnica ed Elettronica, Politecnico di Milano,
Piazza L. da Vinci 32, 20133 Milano, Italy

Received 29 July 1977; revised version received 15 September 1977

Spanning trees, min-max problems, computational complexity, matroids



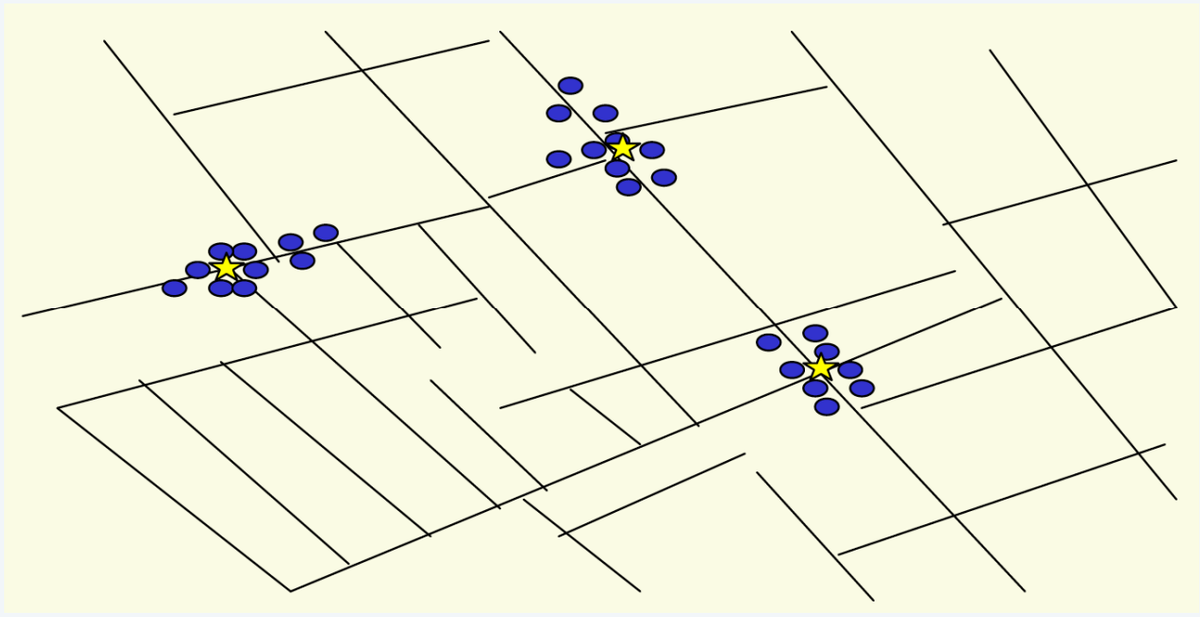
SECTION 4.7

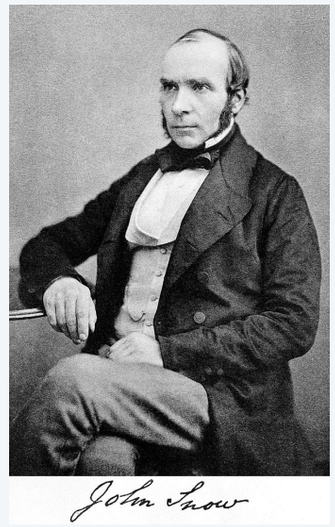
4. ALGORITMI AVIDI II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *clustering a massima spaziatura*
- *min-cost arborescences*

Clustering [raggruppamento]

Scopo. Dato un insieme U di n oggetti etichettati $p_1, \ldots, p_n$, partizionarli in gruppi (cluster) in modo che oggetti di gruppi distinti siano lontani tra loro.





epidemia di morti per colera nella Londra del 1850 (fonte: Nina Mishra)

Applicazioni.

- Instradamento in reti mobili ad-hoc.
- Categorizzazione di documenti per la ricerca web.
- Ricerche di similarità in basi di dati di immagini mediche
- Raggruppamento di corpi celesti in stelle, quasar, galassie
- ...

Clustering a massima spaziatura

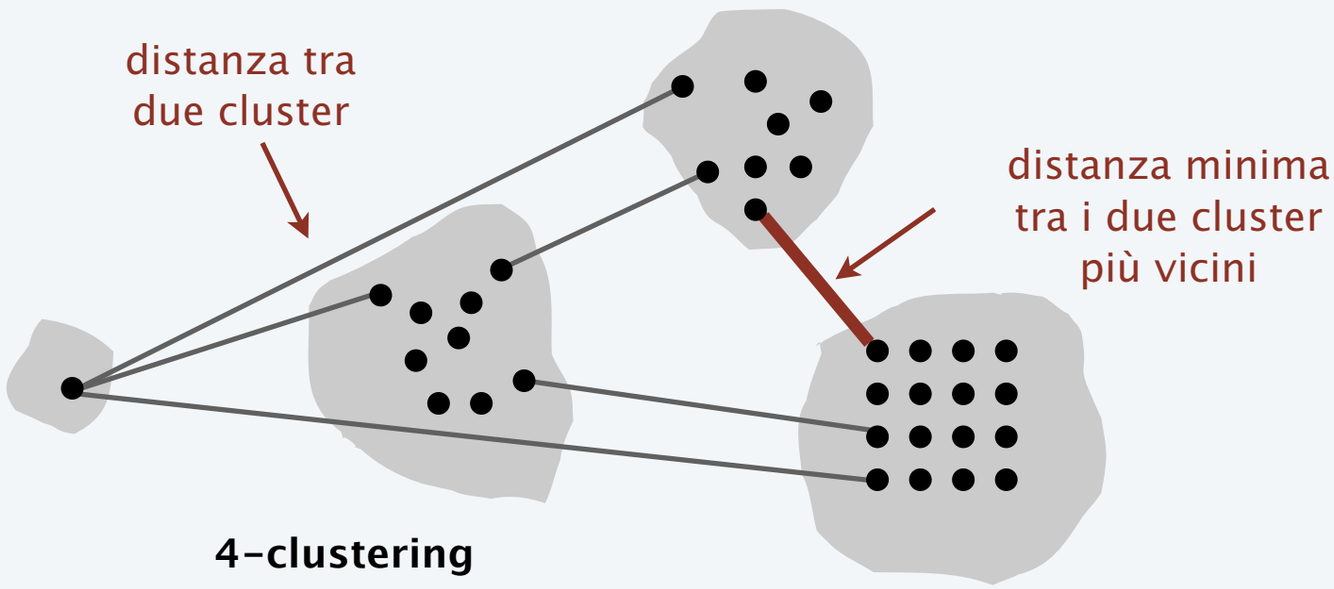
k-clustering. Suddividere gli oggetti in k gruppi non vuoti.

Funzione distanza. Valore numerico per la "vicinanza" tra due oggetti.

- $d(p_i, p_j) = 0$ sse $p_i = p_j$ [identità degli indiscernibili]
- $d(p_i, p_j) \geq 0$ [non-negatività]
- $d(p_i, p_j) = d(p_j, p_i)$ [simmetria]

Spaziatura. Distanza minima tra coppie di punti in cluster distinti.

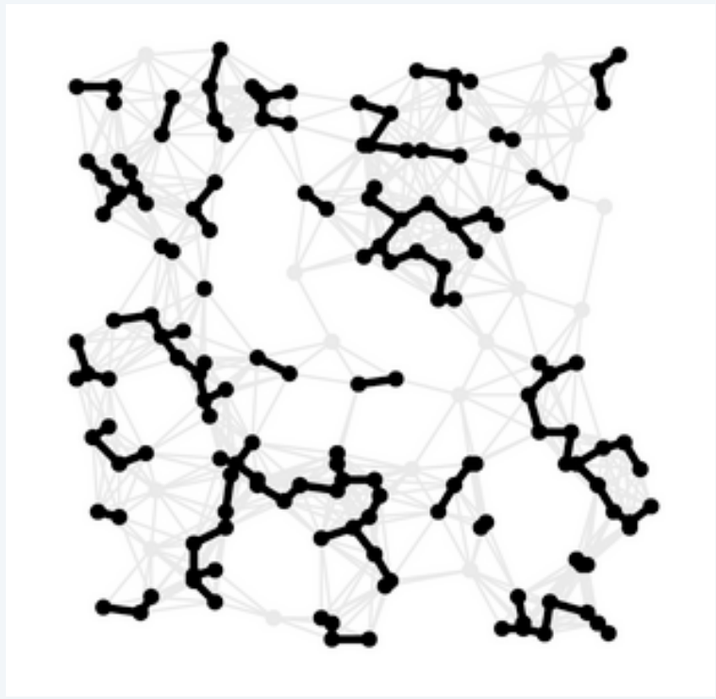
Scopo. Dato un intero k, trovare un k-clustering a massima spaziatura.



Algoritmo avido per il clustering

Algoritmo "ben noto" in letteratura scientifica per il k-clustering:

- Forma un grafo sull'insieme dei nodi U, corrispondenti a n cluster.
- Trova la coppia più vicina di oggetti in cluster diversi, e aggiungi un arco tra i due nodi corrispondenti.
- Ripeti $n - k$ volte (finché non si arriva a k cluster).



Osservazione chiave. Questa procedura è la stessa dell'algoritmo di Kruskal (eccetto che ci si ferma quando si arriva a k componenti connesse).

Alternativa. Trova un MST e cancella i $k - 1$ archi più lunghi.

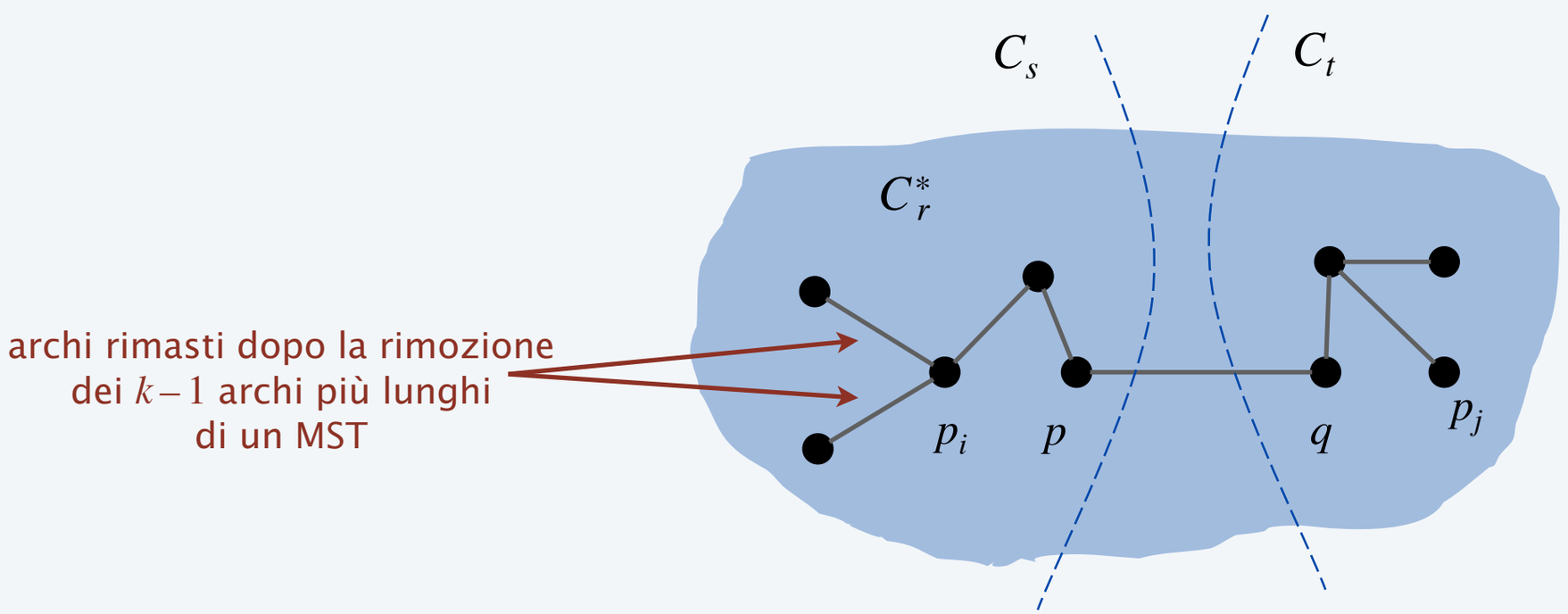
Algoritmo avido per il clustering: analisi

Teorema. Sia C^* un clustering $C_1^*, \ldots, C_k^*$ formato rimuovendo i $k-1$ archi più lunghi di un MST. Allora C^* è un k-clustering a massima spaziatura.

Dim. Sia C un qualunque altro clustering $C_1, \ldots, C_k$.

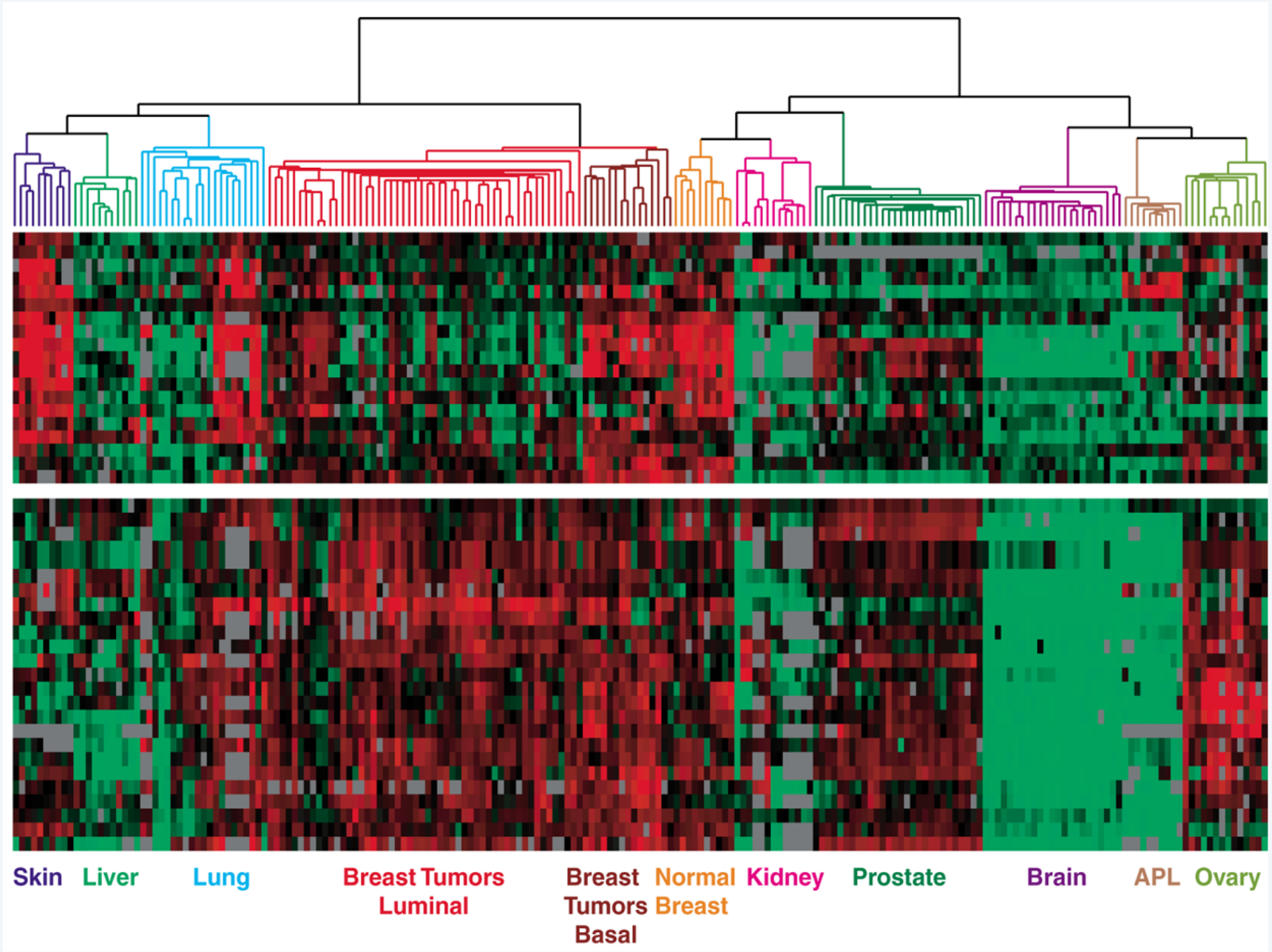
- Siano p_i e p_j nello stesso cluster di C^*, sia esso C_r^*, ma in diversi cluster di C, siano essi C_s e C_t.
- Qualche arco (p, q) sul cammino $p_i - p_j$ in C_r^* interseca due cluster diversi di C.
- Spaziatura di C^*= lunghezza d^* del $(k-1)$esimo arco più lungo di un MST.
- L'arco (p, q) ha lunghezza $\le d^*$ poiché è stato scelto da Kruskal.
- La spaziatura di C è $\le d^*$ poiché p e q sono in cluster diversi. ▪

questo è l'arco che Kruskal avrebbe aggiunto al passo successivo se avessimo proseguito



Dendrogramma dei tumori umani

Tumori in tessuti simili tendono a raggrupparsi assieme nel clustering.



Riferimento: Botstein & Brown group

gene espresso

gene non espresso