

Algoritmi e Strutture Dati

Lezione 23

21 novembre 2022

ALGORITMO Kruskal (Grafo $G = (V, E, \omega)$) \rightarrow albero

ordina E in maniera non decrescente in base ai pesi

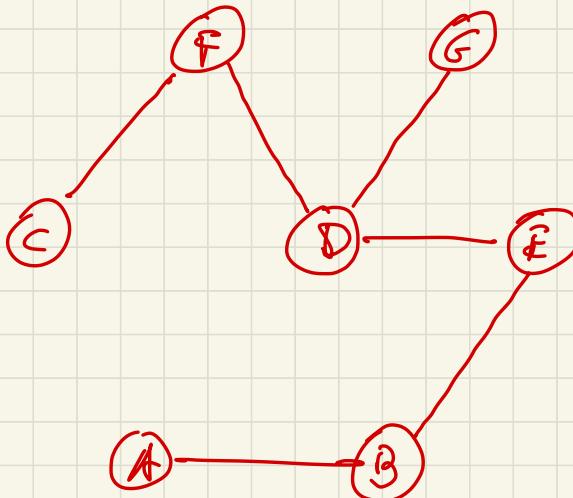
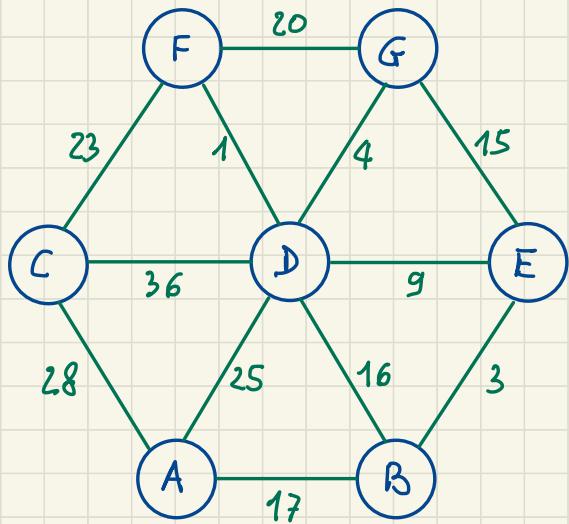
$T \leftarrow (V, \emptyset)$

FOR EACH $(x, y) \in E$ secondo l'ordine DO

| IF x e y non sono connessi in T THEN

| | aggiungi a T l'arco (x, y)

RETURN T



ALGORITMO Kruskal (Grafo $G = (V, E, \omega)$) → albero
 ordina E in maniera non decrescente in base ai pesi
 $T \leftarrow (V, \emptyset)$
 FOR EACH $(x, y) \in E$ secondo l'ordine DO
 IF x e y non sono connessi in T THEN
 L aggiungi a T l'arco (x, y)
 RETURN T

ALGORITMO di KRUSKAL: correttezza

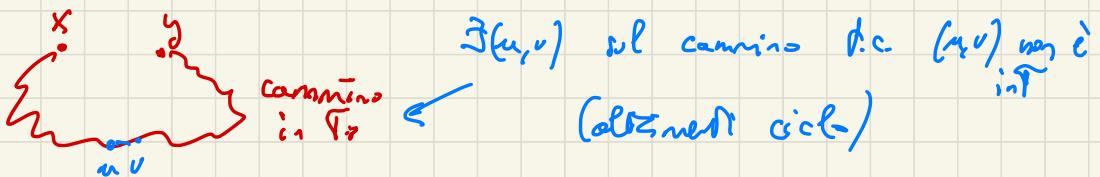
Teo. L'algoritmo di Kruskal trova un albero ricoprente minimo

Dim. T' albero trovato da Kruskal

T' albero ricoprente minimo con il maggior numero di archi in comune con T (\neq)

Per assurdo sia $T_0 \neq T$

Sia (x,y) il primo arco in T' ma non in T_0



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

Modif. T_0 togliendo (u,v) e aggiungendo (x,y)

Ottengo albero ricoprente \bar{T}

$$w(\bar{T}) = w(T_0) - w(u,v) + w(x,y) \leq w(T_0)$$

T_0 minimo $\Rightarrow \bar{T}$ minimo

\bar{T} ha un arco in comune con T in più

di $T_0 \rightarrow$ ASSURDO (\neq)

ALGORITMO Kruskal (Grafo $G = (V, E, \omega)$) \rightarrow albero

ordina E in maniera non decrescente in base ai pesi

$T \leftarrow (V, \emptyset)$

FOR EACH $(x, y) \in E$ secondo l'ordine DO

 | IF x e y non sono connessi in T THEN

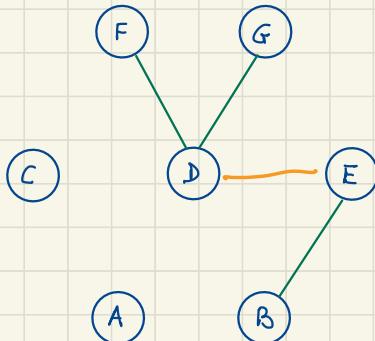
 | aggiungi a T l'arco (x, y)



RETURN T

ALGORITMO di KRUSKAL: implementazione

- Organizziamo l'insieme dei vertici utilizzando una partizione
- Due vertici appartengono allo stesso elemento della partizione se e solo se sono connessi da un cammino



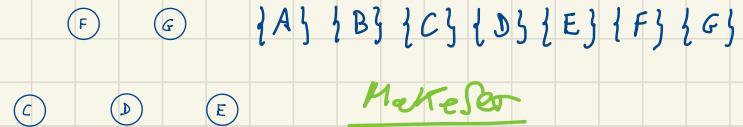
$\{A\} \{B, E\} \{C\} \{D, F, G\}$

D, E sono connessi? $\rightarrow \text{Find}(D) \neq \text{Find}(E) \rightarrow \text{NO}$

G, E sono connessi? $\rightarrow \begin{array}{l} \text{Find}(G) \\ = \\ \text{Find}(E) \end{array} \rightarrow \text{SI}$

ALGORITMO di KRUSKAL: implementazione

• Partizione iniziale: singolari.



• Ad ogni passo:

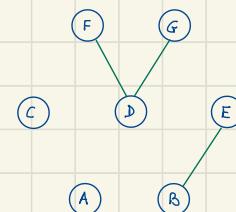
x e y non connessi in T

sse

x e y appartengono a
insiemi differenti



{A} {B,E} {C} {D,F,G}



FIND(x)

FIND(y)

Aggiunta di un arco: unione di due elementi della partizione

UNION (x, y)

• Rappresentazione del grafo: lista di archi

ALGORITMO di KRUSKAL: implementazione UNION/FIND

ALGORITMO Kruskal (Grafo $G = (V, E, \omega)$) → albero

ordina E in maniera non decrescente in base ai pesi

$T \leftarrow (V, \emptyset)$

FOR EACH vertice $v \in V$ DO makeSet(v)

FOR EACH $(x, y) \in E$ secondo l'ordine DO

$T_x \leftarrow \text{FIND}(x)$

$T_y \leftarrow \text{FIND}(y)$

IF $T_x \neq T_y$ THEN

UNION (T_x, T_y)

aggiungi a T l'arco (x, y)

RETURN T

ALGORITMO Kruskal (Grafo $G = (V, E, \omega)$) → albero

ordina E in maniera non decrescente in base ai pesi

$T \leftarrow (V, \emptyset)$

FOR EACH $(x, y) \in E$ secondo l'ordine DO

IF x e y non sono connessi in T THEN

aggiungi a T l'arco (x, y)

RETURN T

ALGORITMO di KRUSKAL: complessità in Tempo $n = \#V$ $m = \#E$

[1] HeapSort

$O(m \lg m)$

[2] for each $v \in V$ MAKESET

$O(n)$

[3] For each (archi) m iterazioni:

[3a] # for find $2m$

$O(m \lg n)$

[3b] # for union $\frac{n-1}{\# archi}$

$O(n)$

$$O(m \lg m) + O(n) + O(m \lg n) + O(n)$$

$$\leq O(m \lg n^2) + O(m \lg n) = O(m \lg n)$$

$$n-1 \leq m \leq n^2$$

$$2m \lg n$$

ALGORITMO Kruskal (Grafo $G = (V, E, w)$) → albero

[4] ordina E in maniera non decrescente in base ai pesi
 $T \leftarrow (V, \emptyset)$

[5] FOR EACH vertice $v \in V$ DO makeSet(v)

[6] FOR EACH $(x, y) \in E$ secondo l'ordine DO

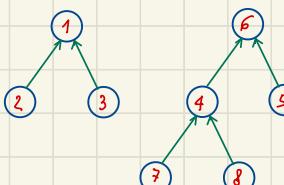
[7] $T_x \leftarrow \text{FIND}(x)$] 2 passi
 $T_y \leftarrow \text{FIND}(y)$] 2 passi

IF $T_x \neq T_y$ THEN

[8] UNION(T_x, T_y)] 1 unione
 $T \leftarrow T \cup \{(x, y)\}$

RETURN T

QuickUnion con bilanciamento in alternanza



MakeSet $O(1)$

Find $O(\lg n)$

Union $O(1)$

ALBERO RICOPRENTE MINIMO:
una strategia greedy alternativa

Problema

Dato $G = (V, E)$ non orientato connesso con una funzione peso $w: E \rightarrow \mathbb{R}$ trovare un albero ricoprente $T = (V, E_T)$ di peso minimo

Inizialmente:

T : albero costituito da un unico vertice

Ad ogni passo:

si espanderà T aggiungendo
l'arco (x, y) di peso minimo
con un vertice in T e l'altro
non in T



ALGORITMO Prim (Grafo $G = (V, E, \omega)$) \rightarrow albero

$T \leftarrow$ albero costituito da un unico vertice se V qualiasi.

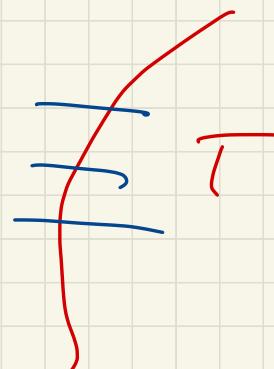
WHILE T ha meno di n vertici DO

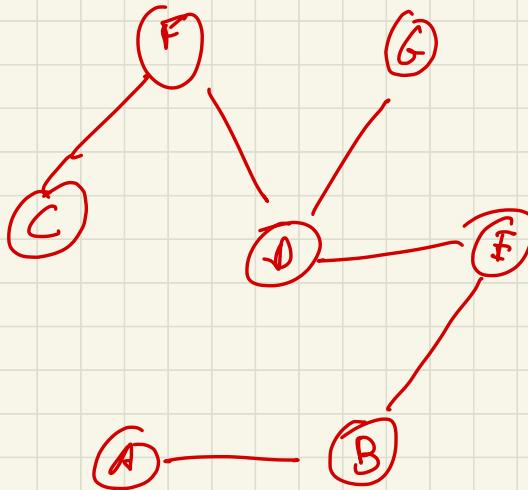
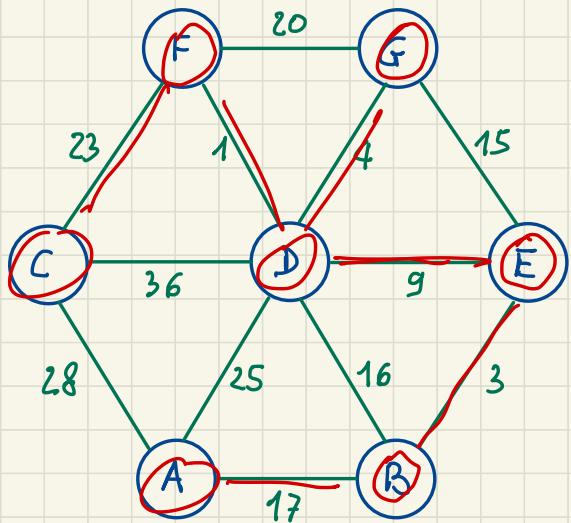
Sia (x, y) l'arco di peso minimo

con $x \in T$ e y non in T

aggiungere a T il vertice y e l'arco (x, y)

RETURN T





ALGORITMO $\text{Prim} (\text{Grafo } G = (V, E, \omega)) \rightarrow \text{albero}$

$T \leftarrow \text{albero costituito da un unico vertice } s \in V \text{ qualsiasi}$

WHILE T ha meno di n nodi DO

sia (x, y) l'arco di peso minimo
con x in T e y non in T

aggiungi a T il vertice y e l'arco (x, y)

RETURN T

ALGORITMO di PRIM : correttezza

Teo. L'algoritmo di Prim trova un albero circonferenza minima

\Rightarrow PRIM

Trova sempre la soluzione
ottima

PRIM vs KRUSKAL

Strategie greedy differenti

- KRUSKAL

- soluzione parziale: foresta di alberi con insieme di vertici V
- inizialmente: no archi, tutti i vertici

- PRIM

- soluzione parziale: albero $T = (V_T, E_T)$ con $V_T \subseteq V$
 $E_T \subseteq E \cap (V_T \times V_T)$
- inizialmente: albero formato da un unico vertice

ALGORITMO di PRIM: implementazione

ALGORITMO Prim (Grafo $G = (V, E, w)$) \rightarrow albero

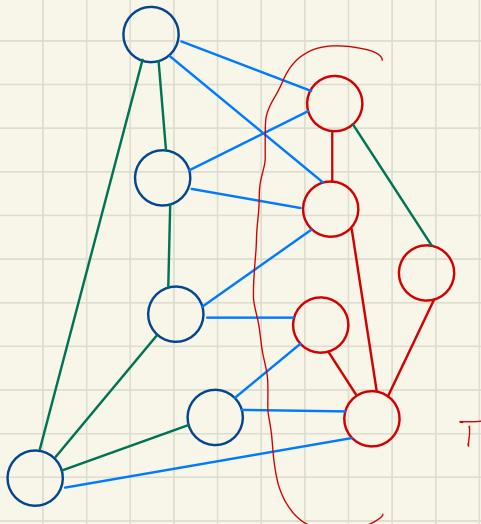
$T \leftarrow$ albero costituito da un unico vertice $s \in V$ qualsiasi;

WHILE T ha meno di n nodi DO

sia (x, y) l'arco di peso minimo
con x in T e y non in T

aggiungere a T il vertice y e l'arco (x, y)

RETURN T



ALGORITMO di PRIM: implementazione

AD OGNI PASSO PER OGNI VERTICE v

NON ANCORA IN T CONSIDERIAMO:

$d[v] = \min$ peso degli archi
tra v e vertici in T
(∞ se non ce ne sono)

$$d[v] = \min \{ \omega(u, v) \mid u \in V_T \} \cup \{ \infty \} \quad T = (V_T, E_T)$$

$\text{vicino}[v] = \text{vertice } u \text{ in } T \text{ t.c. } d[v] = \omega(u, v)$

ALGORITMO Prim (Grafo $G = (V, E, \omega)$) \rightarrow albero

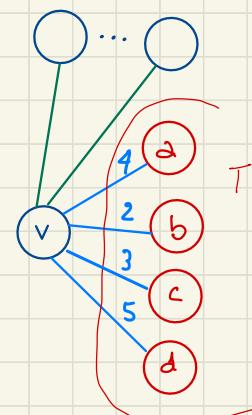
$T \leftarrow$ albero costituito da un unico vertice $s \in V$ qualsiasi

WHILE T ha meno di n nodi DO

sia (x, y) l'arco di peso minimo
con x in T e y non in T

aggiungi: a T il vertice y e l'arco (x, y)

RETURN T

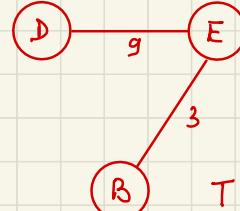
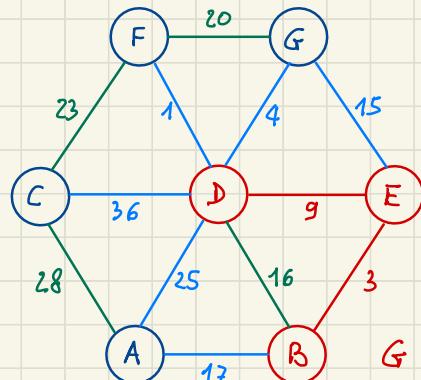


ALGORITMO di PRIM: implementazione

$d[v]$ = minimo peso degli archi
tra v e vertici in T
(∞ se non ce ne sono)

$$d[v] = \min \{ w(u, v) \mid u \in V_T \} \cup \{ \infty \} \quad T = (V_T, E_T)$$

$Vicino[v]$ = vertice u t.c. $d[v] = w(u, v)$



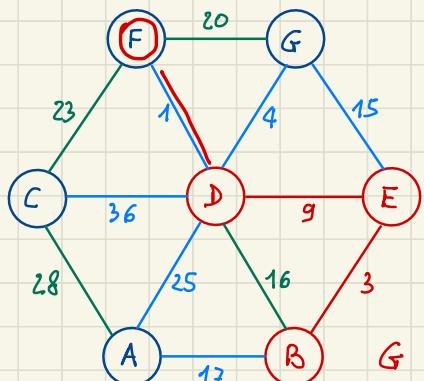
	d	$Vicino$
A	17	B
C	36	D
F	1	D
G	4	D

ALGORITMO di PRIM: implementazione

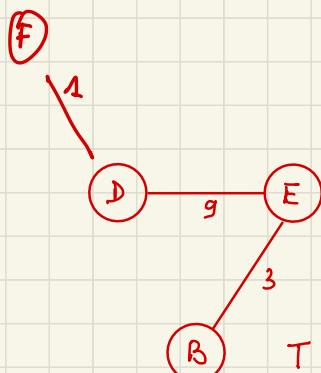
$d[v]$ = minimo peso degli archi
tra v e vertici in T
(∞ se non ce ne sono)

$$d[v] = \min \{ w(u, v) \mid u \in V_T \} \cup \{ \infty \} \quad T = (V_T, E_T)$$

Vicino [v] = vertice u t.c. $d[v] = w(u, v)$



+ Costa con priorità
et vertici con chiavi $d[v]$



ALGORITMO Prim (grafo $G = (V, E, w)$) \rightarrow albero

$T \leftarrow$ albero costituito da un unico vertice $s \in V$ qualsiasi

WHILE T ha meno di n nodi DO

sia (x, y) l'arco di peso minimo
con x in T e y non in T

aggiungi: a T il vertice y e l'arco (x, y)

RETURN T

	d	Vicino
A	17	B
C	23	D, F
F	1	D
G	4	D

ALGORITMO Prim (Grafo $G = (V, E, \omega)$) \rightarrow albero

Sia C una coda con priorità minima

Siano d e vicino due array con indici in V

FOR EACH $v \in V$ DO

$d[v] \leftarrow \infty$

$C.\text{insert}(v, \infty)$

$T \leftarrow (\emptyset, \emptyset)$

DO

$y \leftarrow C.\text{deleteMin}()$

$V_T \leftarrow V_T \cup \{y\}$

IF $d[y] \neq \infty$ THEN

//falsa solo alla prima iterazione

$x \leftarrow \text{vicino}[y]$

$E_T \leftarrow E_T \cup \{(x, y)\}$

FOR EACH $(y, z) \in E$ DO

IF $z \notin V_T$ AND $\omega(y, z) < d[z]$ THEN

$d[z] \leftarrow \omega(y, z)$

$C.\text{changeKey}(z, \omega(y, z))$

$\text{vicino}[z] \leftarrow y$

WHILE $C \neq \emptyset$

RETURN T

ALGORITMO Prim (Grafo $G = (V, E, \omega)$) \rightarrow albero

$T \leftarrow$ albero costituito da un unico vertice $s \in V$ qualsiasi:

WHILE T ha meno di n nodi DO

sia (x, y) l'arco di peso minimo con x in T e y non in T

aggiungi a T il vertice y e l'arco (x, y)

RETURN T

Inizialmente

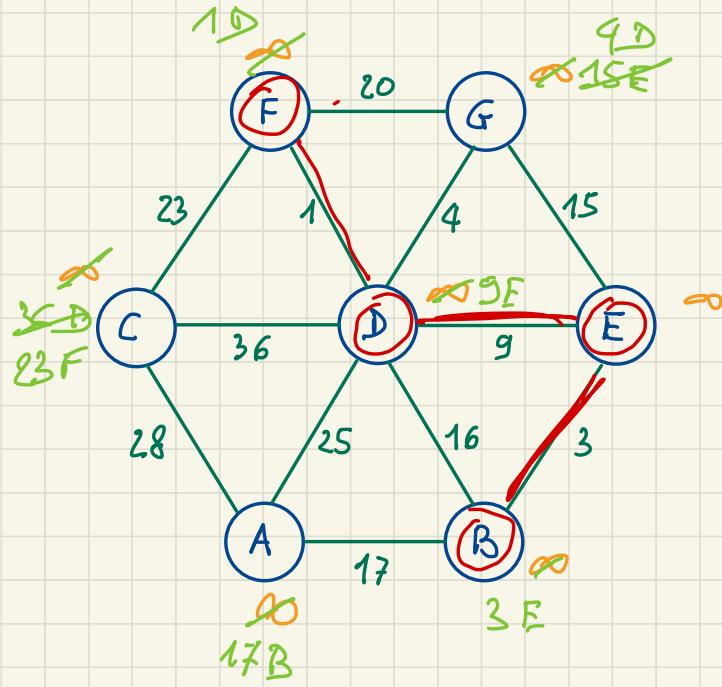
ogni vertice v ha priorità $d[v] = \infty$
(albero vuoto)

Ad ogni passo:

Si preleva vertice y con $d[y]$ minima

Si aggiunge a T il vertice y e l'arco (x, y) con $x = \text{vicino}[y]$

Si aggiorna priorità $d[z]$ e $\text{vicino}[z]$ per ogni vertice z adiacente a y



ALGORITMO Prim (Grafo $G = (V, E, \omega)$) → altera

Sia C una coda con priorità vuota

Siano d e vicino due array con indici in V

FOR EACH $v \in V$ DO

$d[v] \leftarrow \infty$

C.insert(v, 0)

$$T \leftarrow (\emptyset, \emptyset)$$

Pf

$y \leftarrow C.\text{deleteMin}()$

$$V_T \leftarrow V_T \cup \{y\}$$

IF $d[y] \neq \infty$ THEN //falsa solo alla prima
iterazione,

$x \leftarrow \text{vicioho}[y]$

$\leftarrow E_T \cup \{(x, y)\}$

FOR EACH $(y, z) \in E$ DO

IF $z \notin V_T$ AND $\omega(y, z) < d[z]$ THEN

$$d[z] \leftarrow \omega(y, z)$$

C. changeKey(z, w(y,z))

vicino[z] ← y

WHILE $C \neq \emptyset$

RETURN T

ALGORITMO di PRIM: implementazione con coda con priorità

Rappresentazione di G: liste di adiacenza

Coda con priorità C: heap di n elementi
(vettore posizionale)

[1] riempি coda \equiv riempি array

Tempo
 $O(n)$

[2] n iterazioni \Rightarrow WHILE

[2a] dels min \leftarrow coste singole $O(n \log n)$

$O(n \log n)$

[2b] tempo costante \times n volte $O(n)$

[2c] in totale 2n iterazioni

del changeKey al max m \leftarrow coste singole $O(m \log n)$

$$O(n) + O(n \log n) + O(n) + O(m \log n) \\ = O(m \log n)$$

$n-1 \leq m \leq n$

ALGORITMO Prim (grafo $G = (V, E, w)$) \rightarrow albero

Sia C una coda con priorità min

Siano d e vicino due array con indici in V

FOR EACH $v \in V$ DO

$d[v] \leftarrow \infty$
C.insert(v, ∞)

$T \leftarrow (\emptyset, \emptyset)$

DO

$y \leftarrow C.\underline{\text{deleteMin}}()$

$V_T \leftarrow V_T \cup \{y\}$

IF $d[y] \neq \infty$ THEN

$x \leftarrow \text{vicino}[y]$

$E_T \leftarrow E_T \cup \{(x, y)\}$

FOR EACH $(y, z) \in E$ DO

IF $z \notin V_T$ AND $w(y, z) < d[z]$ THEN

$d[z] \leftarrow w(y, z)$

C.changeKey(z, w(y, z))

$\text{vicino}[z] \leftarrow y$

WHILE $C \neq \emptyset$

RETURN T

[2a]

[2b]

[2c]

[2d]

[2e]

$$T(n, m) = O(m \lg n)$$

Heap di Fibonacci

↳ change key cost $O(1)$ ammortizzato

sp la chiave
diminuisce

↳ costo totale $O(m + n \lg n)$