Algoritmi e Strutture Dati

Lezione 26

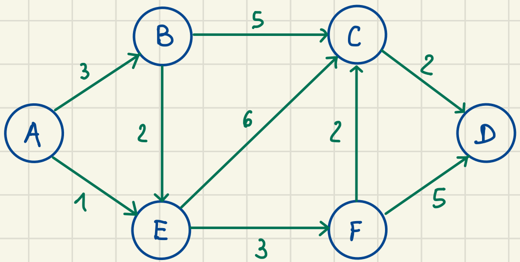
21 novembre 2025

CAMMINI MINIMI DA UN VERTICE A TUTTI GLI ALTRI

L'ALGORITMO DI Dijkstra

INPUT $G = (V, E)$ $\quad \omega : E \to \mathbb{R}$

$s \in V$ vertice di partenza



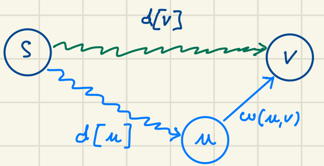
distanze "provvisorie"

$d[v]$ = peso del cammino minimo da s a v sinora trovato

Inizialmente:

$$d[v] = \begin{cases} 0 & \text{se } v = s \\ \infty & \text{altrimenti} \end{cases}$$

Aggiornamento:



$d[u] + \omega(u,v) < d[v]$?

ad ogni passo si considerano gli archi che escono da un vertice u scelto con strategia GREEDY

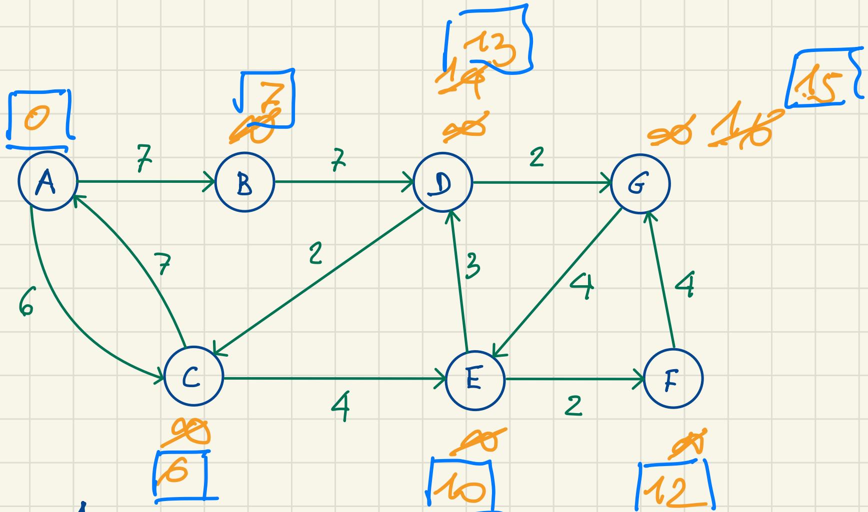
IF $d[u] + \omega(u,v) < d[v]$ THEN

$\quad d[v] \leftarrow d[u] + \omega(u,v)$

L'ALGORITMO DI Dijkstra

INPUT $G = (V, E)$ $\quad \omega: E \to \mathbb{R}$

$s \in V$ vertice di partenza



$s = A$

- Distanze provvisorie vettore $d[V]$

  inizialmente $d[v] = \begin{cases} 0 & \text{se } v = s \\ \infty & \text{se } v \neq s \end{cases}$

- $C \subseteq V$ insieme vertici candidati

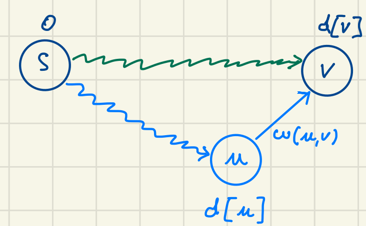
  inizialmente $C = V$

- strategia "greedy"

  preleva da C il vertice u con $d[u]$ minima

  $d[u]$ diventa definitiva

  aggiorna $d[v]$ per ogni v adiacente a u



$d[u] + \omega(u,v) < d[v]$?

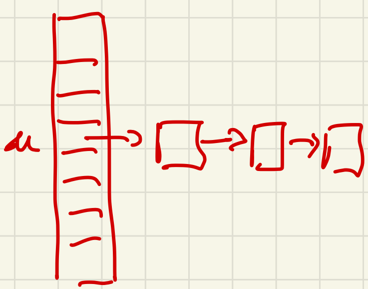
ALGORITMO di DIJKSTRA: implementazione

```
ALGORITMO Dijkstra (Grafo G, vertice s) → Vettore
  Sia d[V] un vettore con indici in V
  d[s] ← 0
  FOR EACH v ∈ V \ {s} DO d[v] ← ∞
  C ← V
  WHILE C ≠ ∅ DO
    u ← elemento di C con d[u] minima
    C ← C \ {u}
    FOR EACH (u,v) ∈ E DO
      IF d[u] + w(u,v) < d[v] THEN
        d[v] ← d[u] + w(u,v)
  RETURN d
```

lista di adiacenza →

lista di incidenza

← m passi complessivi



ALGORITMO Dijkstra (Grafo G, vertice s) $\rightarrow$ Vettore

Sia $d[V]$ un vettore con indici in V

Sia C una coda con priorità vuota

$d[s] \leftarrow 0$

FOR EACH $v \in V \setminus \{s\}$ DO $d[v] \leftarrow \infty$

FOR EACH $v \in V$ DO $C.\mathrm{insert}(v, d[v])$

~~$C \leftarrow V$~~

WHILE $C \neq \emptyset$ DO

- ~~$u \leftarrow$ elemento di C con $d[u]$ minima~~
- ~~$C \leftarrow C \setminus \{u\}$~~
- $u \leftarrow C.\mathrm{deleteMin}()$
- FOR EACH $(u,v) \in E$ DO
  - IF $d[u] + w(u,v) < d[v]$ THEN
    - $d[v] \leftarrow d[u] + w(u,v)$, $C.\mathrm{changeKey}(v, d[v])$

RETURN d

ALGORITMO di DIJKSTRA: tempo di calcolo

grafo: lista di adiacenza
coda con priorità
↳ min-heap
↳ vettore posizione

ALGORITMO Dijkstra (Grafo G, vertice s) → Vettore

a
Sia $d[V]$ un vettore con indici in V
$d[s] \leftarrow 0$
FOR EACH $v \in V \setminus \{s\}$ DO $d[v] \leftarrow \infty$

b
Sia C una coda con priorità vuota
FOR EACH $v \in V$ DO $C.\text{insert}(v, d[v])$

c
WHILE $C \neq \emptyset$ DO
    $u \leftarrow C.\text{deleteMin}()$ c1
    FOR EACH $(u,v) \in E$ DO c2
        IF $d[u] + w(u,v) < d[v]$ THEN
            $d[v] \leftarrow d[u] + w(u,v)$
            $C.\text{changeKey}(v, d[v])$ c3
RETURN d

a inizializzazione di d — $O(n)$

b riempimento di C — $O(n)$

c ciclo while
n iterazioni

c1 deletemin $O(\log n)$ costo singolo — $O(n \log n)$

c2 intestazione for each
m iterazioni complessivamente
(lista adiacenza) — $O(m)$

c3 al max m volte
changekey $O(\log n)$ — $O(m \log n)$

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

grafo connesso $m \geq n-1$

$$= O(m \log n) \Leftarrow \text{Tempo}$$

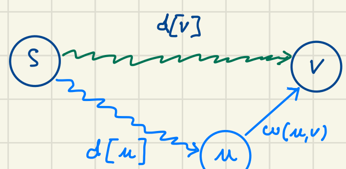
$$\text{Tempo}(n, m) = O(m \log n)$$

usando heap di Fibonacci:

$$\text{Tempo}(n, m) = O(m + n \log n)$$

ALGORITMO di DIJKSTRA: come ricavare i cammini minimi?

vettore dei predecessori: pred[V]



ALGORITMO Dijkstra (Grafo G, vertice s) → Vettore

Sia d[V] un vettore con indici in V

Sia pred[V] un vettore con indici in V

$d[s] \leftarrow 0$

FOR EACH $v \in V \setminus \{s\}$ DO $d[v] \leftarrow \infty$

Sia C una coda con priorità vuota

FOR EACH $v \in V$ DO C.insert(v, d[v])

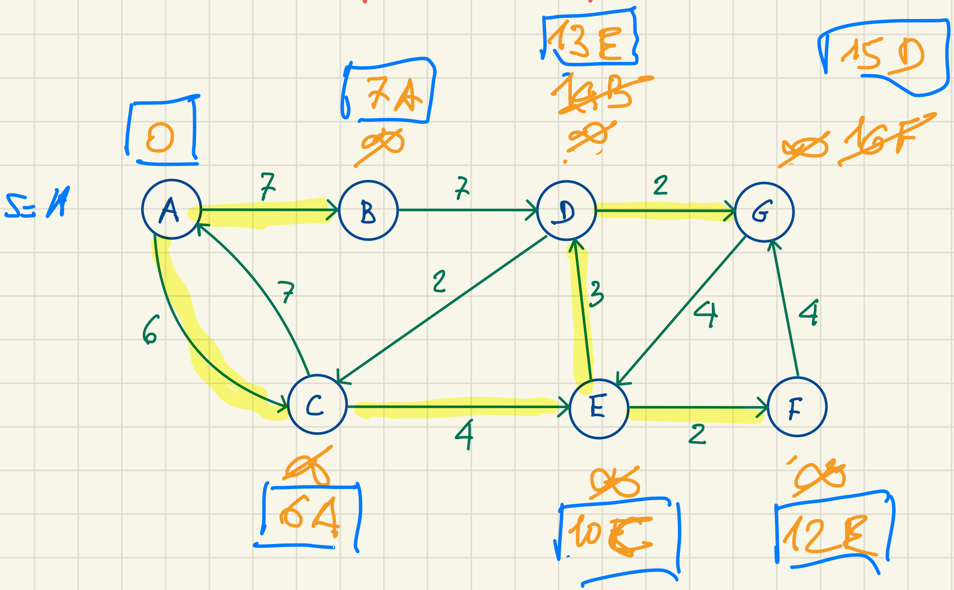
WHILE $C \neq \emptyset$ DO

- $u \leftarrow$ C.deleteMin()
- FOR EACH $(u,v) \in E$ DO
  - IF $d[u] + \omega(u,v) < d[v]$ THEN
    - $d[v] \leftarrow d[u] + \omega(u,v)$
    - C.changeKey(v, d[v])
    - $pred[v] \leftarrow u$

RETURN d

ALGORITMO di DIJKSTRA: come ricavare i cammini minimi?

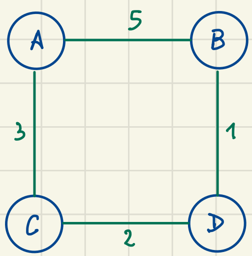
vettore dei predecessori: pred[V]

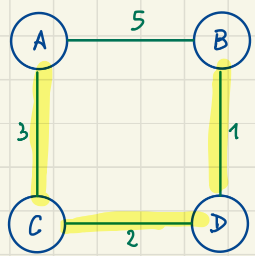


Albero dei cammini minimi

```
ALGORITMO Dijkstra (Grafo G, vertice s) → Vettore, vettore
  Sia d[V] un vettore con indici in V
  Sia pred[V] un vettore con indici in V
  d[s] ← 0
  FOR EACH v ∈ V \ {s} DO d[v] ← ∞
  Sia C una coda con priorità vuota
  FOR EACH v ∈ V DO C.insert(v, d[v])
  WHILE C ≠ ∅ DO
    u ← C.deleteMin()
    FOR EACH (u,v) ∈ E DO
      IF d[u] + ω(u,v) < d[v] THEN
        d[v] ← d[u] + ω(u,v)
        C.changeKey(v, d[v])
        pred[v] ← u
  RETURN d, pred
```

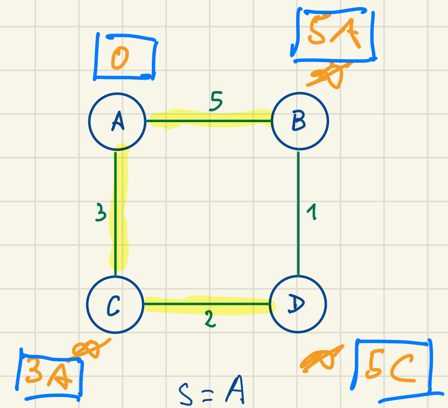
GRAFI NON ORIENTATI: albero ricoprente minimo vs albero dei cammini minimi





Albero ricoprente minimo

peso 6

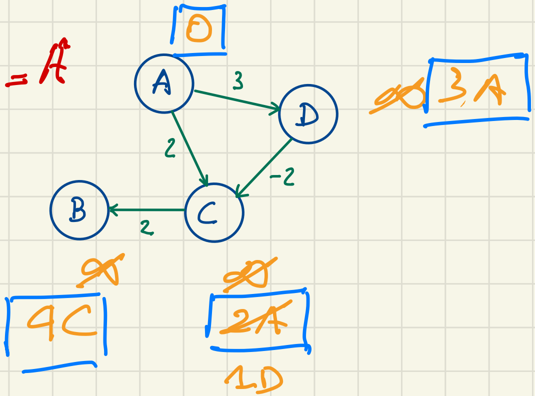


Albero dei cammini minimi da A

peso 10

PESI NEGATIVI

$s = A$



$A \xrightarrow{3} D \xrightarrow{-2} C \xrightarrow{2} B$

3

PROBLEMI "CAMMINI MINIMI"

- Cammino minimo tra due vertici

  non c'è algoritmo diretto

- Cammini minimi da un vertice a tutti gli altri

  Dijkstra

  Tempo $O(m \log n)$

  NO ARCHI PESO NEGATIVO

  Bellman Ford

  $O(m \cdot n)$

  NO CICLI NEGATIVI

- Cammini minimi tra ogni coppia di vertici

  Floyd Warshall

  $O(n^3)$

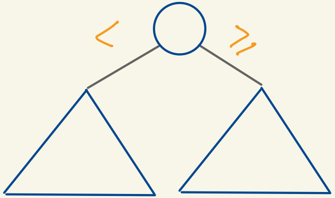
  NO CICLI NEGATIVI

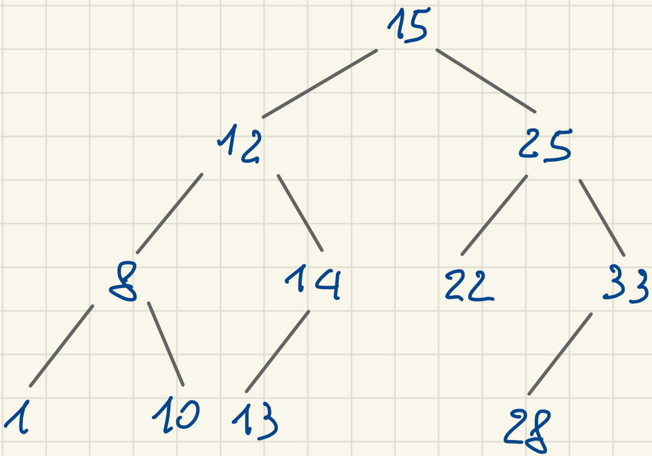
Alberi di ricerca

Alberi binari di ricerca

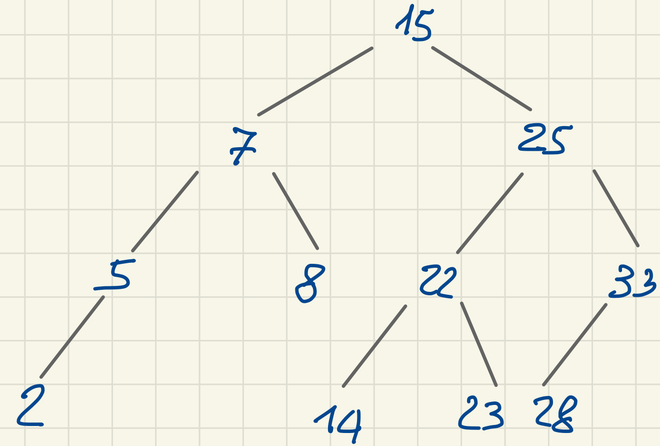
Un *albero binario di ricerca* è un albero binario in cui *per ogni nodo n*:

- il valore di ogni chiave contenuta nel sottoalbero sinistro di n è minore della chiave contenuta in n,
- il valore di ogni chiave contenuta nel sottoalbero destro di n è maggiore o uguale della chiave contenuta in n.

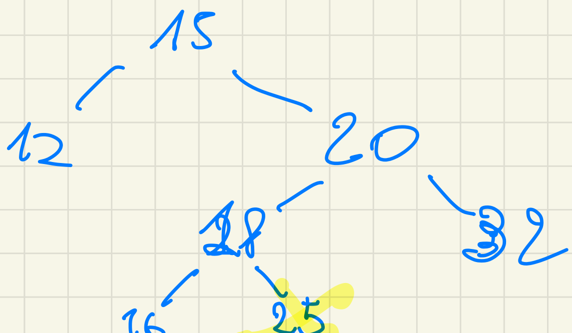


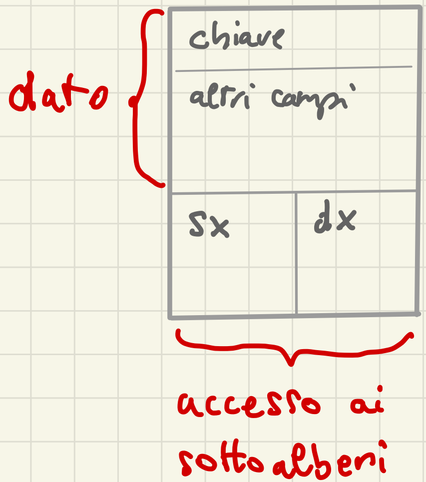

15
12
25
8
14
22
33
1
10
13
28

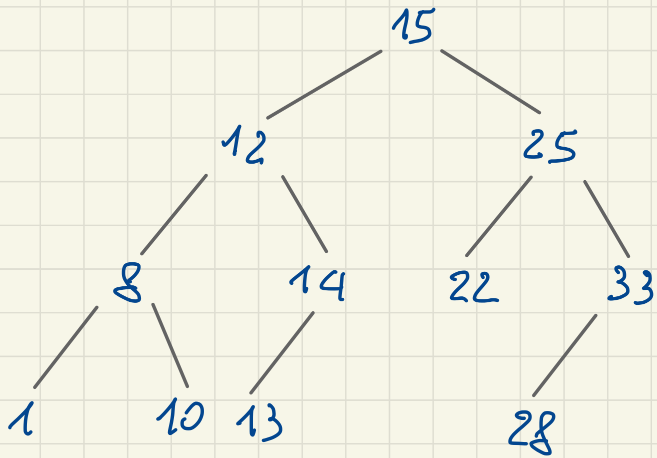
SI


15
7
25
5
8
22
33
2
14
23
28

NO


15
12
20
18
32
16
25


chiave
altri campi
dato
sx
dx
accesso ai sottoalberi



visita in ampiezza

15 12 25 8 14 22 33

1 10 13 28

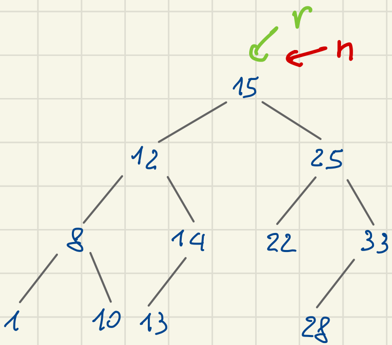
visita in profondità in ordine simmetrico

SX - Radice - dx

< ≥

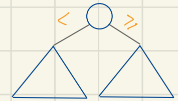
1 8 10 12 13 14 15 22 25 28 33

Trovare il nodo con chiave max (o min)




```
FUNZIONE massimo (AlberoRicerca r) → Nodo
    IF r = null THEN
        RETURN null
    ELSE
        n ← r
        WHILE n.dx ≠ null DO
            n ← n.dx
        RETURN n
```

Ricerca ricorsiva




```
FUNZIONE trova (AlberoRicerca r, TipoChiave k) → Nodo
    IF r = null THEN
        RETURN null
    ELSE IF k < r.chiave THEN
        RETURN trova (r.sx, k)
    ELSE IF k > r.chiave THEN
        RETURN trova (r.dx, k)
    ELSE RETURN r
```

ricorsioni in coda