

ALGORITMI + ESEMPI

BFS (V, E)

for ogni vertice $u \in V[G] - \{s\}$

color [u] \leftarrow white

$\pi[u] \leftarrow \text{NIL}$

$d[u] \leftarrow +\infty$

color [s] \leftarrow GREY

$d[s] \leftarrow 0$

$\pi[s] \leftarrow \text{NIL}$

$Q \leftarrow \emptyset$

ENQUEUE (Q, s)

while ($Q \neq \emptyset$)

$u \leftarrow \text{DEQUEUE}(Q)$

for ogni $v \in \text{Adj}[u]$

if color [v] = white

then color [v] \leftarrow GREY

$\pi[v] \leftarrow u$

$d[v] \leftarrow d[u] + 1$

ENQUEUE (Q, v)

color [u] \leftarrow BLACK

DFS(G)

for each $u \in V[G]$

$\text{color}[u] \leftarrow \text{WHITE}$

$\pi[u] \leftarrow \text{NIL}$

$\text{time} \leftarrow 0$

for each $u \in V[G]$

if $\text{color}[u] = \text{WHITE}$

DFS-Visit(u)

DFS-Visit(u)

$\text{time} \leftarrow \text{time} + 1$

$d[u] \leftarrow \text{time}$

$\text{color}[u] \leftarrow \text{GREY}$

for each $v \in \text{Adj}[u]$

if $\text{color}[v] = \text{WHITE}$

$\pi[v] \leftarrow u$

DFS-Visit(v)

$\text{color}[u] \leftarrow \text{BLACK}$

$\text{time} \leftarrow \text{time} + 1$

$f[u] \leftarrow \text{time}$

KRUSKAL

$A \leftarrow \emptyset$

for each $v \in V[G]$

do MAKE-SET(v)

Sort(E)

// Ordina gli archi di E per peso w NON DECRESCENTE

for ogni arco $(u, v) \in E[G]$

if FIND-SET(u) \neq FIND-SET(v)

$A \leftarrow A \cup \{u, v\}$

Union(u, v)

return A

UNION(x, y)

LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)

if $x.rank > y.rank$

$y.p \leftarrow x$

else $x.p \leftarrow y$

if $x.rank = y.rank$

$y.rank = y.rank + 1$

FIND-SET(x)

if $x \neq x.p$

$x.p \leftarrow \text{FIND-SET}(x.p)$

return $x.p$

MAKE-SET(x)

$x.rank \leftarrow 0$

$x.p \leftarrow x$

MST_PRIM(G, w, r)

$Q \leftarrow V[G]$

for ogni $u \in V[G] - \{r\}$

$\text{key}[u] \leftarrow +\infty$

$\text{key}[r] \leftarrow 0$

$\pi[r] \leftarrow \text{NIL}$

while ($Q \neq \emptyset$)

$u \leftarrow \text{extractMin}(Q)$

for ogni $v \in \text{Adj}[u]$

if $v \in Q$ and $w(u, v) < \text{key}[v]$

$\pi[v] \leftarrow u$

$\text{key}[v] \leftarrow w(u, v)$

DJIKSTRA

INITIALIZE-SINGLE-SOURCE (G, s)

for ogni $v \in V[G] - \{s\}$

$$d[v] \leftarrow \infty$$

$$\pi[v] \leftarrow \text{NIL}$$

$$d[s] \leftarrow 0$$

$$\pi[s] \leftarrow \text{NIL}$$

RELAX (u, v, w)

if $d[v] > d[u] + w(u, v)$

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

$$\pi[v] \leftarrow u$$

DJIKSTRA (G, w, s)

INITIALIZE-SINGLE-SOURCE (G, s)

$$Q \leftarrow V[G]$$

$$S \leftarrow \emptyset$$

while ($Q \neq \emptyset$)

$$u \leftarrow \text{extractMin}(Q)$$

$$S \leftarrow S \cup \{u\}$$

for ogni $v \in \text{Adj}[u]$

$$\text{RELAX}(u, v, w)$$

BELLMAN-FORD (G, w, s)

INITIALIZE-SINGLE-SOURCE (G, s)

for $i \leftarrow 1$ to $|V| - 1$

for ogni arco $(u, v) \in E[G]$
 RELCAX(u, v, w)

for ogni arco $(u, v) \in E[G]$

 if $d[v] > d[u] + w(u, v)$
 return FALSE

return TRUE

KNAPSACK 01 (w_i, v_i, W)

$n = \text{length}[v]$

for $i = 0$ to n

$v[i, 0] = 0$

for $w = 0$ to W

$v[0, w] = 0$

for $i = 1$ to n

for $w = 1$ to W

if $w < w[i]$

$v[i, w] = v[i-1, w]$

else if $v[i-1, w] > v[i-1, w - w[i]] + v[i]$

$v[i, w] = v[i-1, w]$

else

$v[i, w] = v[i-1, w - w[i]] + v[i]$

return v

LCS. LENGTH (x, y)

$m = x.length$

$n = y.length$

Siano $b[1..m, 1..n] \in c[0..m, 0..n]$ due nuove tabelle
for $i = 1$ to m

$c[i, 0] = 0$

for $j = 0$ to n

$c[0, j] = 0$

for $i = 1$ to m

for $j = 1$ to n

if $x_i = y_j$

$c[i, j] = c[i-1, j-1] + 1$

$b[i, j] = "R"$

else if $c[i-1, j] \geq c[i, j-1]$

$c[i, j] = c[i-1, j]$

$b[i, j] = "↑"$

else

$c[i, j] = c[i, j-1]$

$b[i, j] = "←"$

return $b \cdot c$

PRINT-LCS(b, x, i, j)

if $i == 0$ OR $j == 0$

return

if $b[i, j] = "R"$

print-LCS(b, x, i-1, j-1)

print X;

else if $b[i, j] = "U"$

print-LCS(b, x, i-1, j)

else

print-LCS(b, x, i, j-1)

FRACTIONAL KNAPSACK (V , W , X , TOT)

for $k \leftarrow 1$ to N

$x[k] \leftarrow 0.0$

Value $\leftarrow 0.0$

Capacity $\leftarrow TOT$

$k \leftarrow 1$

while $k \leq N$ and Capacity > 0.0

if $W[k] \leq Capacity$

 Value $\leftarrow Value + V[k]$

 Capacity $\leftarrow Capacity - W[k]$

$x[k] \leftarrow 1.0$

else

$x[k] \leftarrow Capacity / W[k]$

 Value $\leftarrow Value + x[k] * V[k]$

 Capacity $\leftarrow 0.0$

$k \leftarrow k + 1$

Es

Dato un GRAFO CONNESSO NON ORIENTATO $G(V, E)$ e un vertice $r \in V[G]$, restituire il livello dell'MST radicato in r con più nodi.

MST-PRIM (G, w, r)

$Q \leftarrow V[G]$

for each $u \in V[G] - \{r\}$

$\text{key}[u] \leftarrow +\infty$

$\text{key}[r] \leftarrow 0$

$\pi[r] \leftarrow \text{NIL}$

while ($Q \neq \emptyset$)

$u \leftarrow \text{extractMin}(Q)$

 for each $v \in \text{Adj}[u]$

 if $v \in Q$ and $w(u, v) < \text{key}[v]$

$\pi[v] \leftarrow u$

$\text{key}[v] \leftarrow w(u, v)$

BFS (G^{MST}, r)

$\max \leftarrow 0$

$\text{max-level} \leftarrow 0$

$\text{count}[] \leftarrow \emptyset$

for each $u \in V[G] - \{r\}$

$\text{color}[u] \leftarrow \text{WHITE}$

$\pi[u] \leftarrow \text{NIL}$

$dl[u] \leftarrow +\infty$

$\text{color}[r] \leftarrow \text{GREY}$

$\pi[r] \leftarrow \text{NIL}$

$d[r] \leftarrow 0$

$Q \leftarrow \emptyset$

ENQUEUE (Q, r)

while ($Q \neq \emptyset$)

$u \leftarrow \text{DEQUEUE} (Q)$

for each $v \in \text{Adj}[u]$

if $\text{color}[v] = \text{WHITE}$

then $\text{color}[v] \leftarrow \text{GREY}$

$\pi[v] \leftarrow u$

$d[v] \leftarrow d[u] + 1$

$\text{count}[d[v]] \leftarrow +$

if ($\text{count}[d[v]] > \max$)

$\max \leftarrow \text{count}[d[v]]$

$\max_level \leftarrow d[v]$

ENQUEUE (Q, v)

$\text{color}[u] \leftarrow \text{BLACK}$

return (\max, \max_level)

Es.

Dato un grafo ORIENTATO PESATO $G(V, E)$ con funzione peso $w: E \rightarrow \mathbb{R}^+$ = un peso k , RESTITUIRE una lista di nodi che hanno k come peso del CAMMINO MINIMO da un NODO sorgente s .

DJIKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE (G, s)

$Q \leftarrow V[G]$

$S \leftarrow \emptyset$

$A \leftarrow \emptyset$

while ($Q \neq \emptyset$)

$u \leftarrow \text{extractMin}(Q)$

if $d[u] == k$

$A \leftarrow A \cup \{u\}$

$S \leftarrow S \cup \{u\}$

for ogni $v \in \text{Adj}[u]$

RELAX(u, v, w)

return A

Es.

Dato un GRAFO ORIENTATO con ARCHI di PESO UNITARIO, trovare il LOOP più GRANDE.

DFS (G)

$\max \leftarrow 0 \quad B \leftarrow \emptyset$

for ogni $v \in V[G]$

color [v] \leftarrow WHITE

$\pi[v] \leftarrow \text{NIL}$

time $\leftarrow 0$

for ogni $v \in V[G]$

if color [v] = WHITE

DFS.Visit (G, v, \max, B)

DFS.Visit (G, v, \max, B)

time \leftarrow time + 1

d [u] \leftarrow time

color [u] \leftarrow GREY

$A \leftarrow \emptyset$

for ogni $v \in \text{Adj}[u]$

if color [v] = WHITE

$\pi[v] \leftarrow u$

DFS.Visit (G, v, \max, B)

else if color [v] = GREY

$\pi[v] \leftarrow u$

START $\leftarrow u$

END $\leftarrow v$

COUNT $\leftarrow 1$

while ($\text{START} \neq \text{END}$)
 $A \leftarrow A \cup \{ \text{START} \}$
 $\text{COUNT}++$
 $\text{START} \leftarrow \pi[\text{START}]$
 $A \leftarrow A \cup \{ \text{END} \}$
if ($\text{COUNT} > \text{max}$)
 $\text{max} \leftarrow \text{COUNT}$
 $B \leftarrow A$
 $A \leftarrow \emptyset$

$\text{color}[u] \leftarrow \text{BLACK}$
 $\text{time} \leftarrow \text{time} + 1$
 $f[u] \leftarrow \text{time}$

Es.

8. Dato un grafo orientato di peso unitario, restituire true se al suo interno è presente un ciclo, false altrimenti. Restituire la lista dei nodi che fanno parte del ciclo. Oppure restituire TRUE, start e end e fare il while in DFS.

DFS(G) {

A = \emptyset

$\forall u \in G.V \{$

u.color = WHITE

u. π = NIL

}

time = 0

$\forall u \in G.V \{$

if u.color == WHITE {

(A, flag) = DFS_VISIT(G, u)

}

if flag == TRUE { return (A, flag) }

}

return FALSE

}

DFS_VISIT(G, u) {

u.d = ++time

u.color = GRAY

A = \emptyset

flag = FALSE

$\forall v \in G.adj[u] \{$

if flag == TRUE {

return { A, flag }

}

if v.color == WHITE

v. π = u

(A, flag) = DFS_VISIT(G, v)

} else if v.color == GRAY {

v. π = u

start = u

end = v

while (start != end) {

A = A U { start }

start = start. π

}

A = A U { end }

return (A, TRUE)

}

}

u.color = BLACK

u.f = ++time

return { A, flag }

}

RISCRITTO:

DFS(G)

$A \leftarrow \emptyset$

for ogni $u \in V[G]$

color[u] \leftarrow WHITE

$\pi[u] \leftarrow \text{NIL}$

time $\leftarrow 0$

for ogni $u \in V[G]$

if color[u] = WHITE

$(A, \text{flag}) \leftarrow \text{DFS-Visit}(G, u)$

if flag = TRUE

return (A, flag)

return FALSE

DFS-Visit(G, u)

$A \leftarrow \emptyset$

flag \leftarrow FALSE

time \leftarrow time + 1

$d[u] \leftarrow$ time

color[u] \leftarrow GREY

for ogni $v \in \text{Adj}[u]$

if flag = TRUE

return (A, flag)

if color[v] = WHITE

$\pi[v] \leftarrow u$

$(A, \text{flag}) \leftarrow \text{DFS-Visit}(G, v)$

else if $\text{color}[v] = \text{GREY}$

$\pi[v] \leftarrow u$

$\text{START} \leftarrow u$

$\text{END} \leftarrow v$

while ($\text{START} \neq \text{END}$)

$A \leftarrow A \cup \{ \text{START} \}$

$\text{START} \leftarrow \pi[\text{START}]$

$A \leftarrow A \cup \{ \text{END} \}$

return (A, TRUE)

$\text{color}[u] \leftarrow \text{BLACK}$

$\text{time} \leftarrow \text{time} + 1$

$f[u] \leftarrow \text{time}$

return (A, flag)

ES.

Dato un grafo non orientato G una sorgente s e un intero k , scrivere un algoritmo che calcolato il cammino minimo, restituisca il numero di nodi che si trovano a distanza maggiore di k

BFS(G, k)

Int numeroNodi=0 •

for each $v \in V[G]$

 color[u] = white

 p[u] = NIL

 d[u] = infinito

color[s] = gray

p[s] = NIL

d[s] = 0

Q = lista vuota

enqueue(Q,s)

while (Q diverso da 0)

 u = dequeue(Q)

 for each $v \in \text{Adj}[u]$

 if color[v] = white

 color[v] = gray

 p[v] = u

 dist[v] = dist[u] + 1

 if dist[v] > k

 numeroNodi ++

 enqueue (Q,v)

 color[u] = black

return numeroNodi

STANDARD
BFS

NUOVO

Es.

Dato un grafo orientato e non pesato, scrivere un algoritmo che restituisca true se esiste un arco all'indietro , altrimenti false. Determinare il caso migliore e il caso peggiore.

DFS(G)

for each $u \in V[g]$

 color $[u] = \text{white}$

$p[u] = \text{NIL}$

 time = 0

for each $u \in V[G]$

 if color[u] = white

 ret = DFS_VISIT(u)

 if ret = true

 break

return ret

}

bool DFS_VISIT(u)

color[u] = gray

$d[u] = \text{time} = \text{time} + 1$

for each $v \in \text{Adj}[u]$

 if color[v] = white

 DFS_VISIT(v)

 Else if color[v] = gray

 Return true

T

color[u] = black

$f[u] = \text{time} = \text{time} + 1$

return false

Es.

Dato un grafo non orientato G , una sorgente s e un vertice v , trovare il cammino minimo

```
PercorsoMinimo(G,s,v){  
    BFS(G,s)  
    StampaCammino(v, p)  
}
```

}] NUOVO

```
BFS(G,s)  
for each  $u \in V[G]$   
    color[u] = white  
    p[u]= NIL
```

```
d[u]= infinito  
color[s] = gray  
p[s]=NIL  
d[s]=0
```

```
Q = Lista vuota  
Enqueue(Q,s)
```

While (Q diverso da vuoto)

```
    u = dequeue(Q)  
    for each  $v \in \text{adj}[u]$   
        if color[v]==white  
            color[v]= gray  
            p[v] = u  
            d[v] = d[u]+1  
            enqueue (Q,v)
```

```
    color[u]=black
```

```
StampaCammino(v){  
If(p[v] diverso da NIL)  
    StampaCammino(p[v])  
print v  
}
```

}] NUOVO

STANDARD
BFS

Es.

Dato un grafo connesso, orientato e pesato, con radice nel nodo r , determinare l'altezza dell'MST

Prim(G, w, r)

Altezza = 0

for each $v \in V[G]$

 key[v] = infinito

$p[v] = NIL$

key[r] = 0

Q = $V[G]$

while Q diverso da vuoto

$u = extractmin(Q)$

 for each $v \in Adj[u]$

 if $v \in Q$ AND $w(u,v) < key[v]$

$p[v] = u$

 key[v] = $w(u,v)$

STANDARD
PRIM

altezza += 1

return altezza

NUOVO

Es.

Dato un grafo orientato e pesato con pesi w come numeri reali positivi restituire il numero di nodi con archi uscenti $> k$

INITIALIZE_SINGLE_SOURCE(G, s)

$\text{Lista}[] = \text{vuoto}$

$\text{contaArchi} = 0$] **Nuovo**

$S = \text{insieme vuoto}$

$Q = V[G]$

while Q diverso da insieme vuoto

$u = \text{EXTRACT_MIN}(Q)$

$S = S \cup \{u\}$

**STANDARD
DIJKSTRA**

for ogni vertice $v \in \text{Adj}[u]$

lista[i]++

if lista[i] > k

contaArchi++

RELAX(u, v, w)

+ AGGIUNTE

Es.

15. Dato un grafo orientato e pesato con pesi w come numeri reali positivi restituire il numero di nodi con archi uscenti > k

```
nOutDegree(G) {  
    count = 0  max = -inf      node = nil      A = Ø  
    for i = 0 to |G.V| {  
        for j = 0 to |G.V| {  
            if G[i, j] == 1 {  
                count++  
            }  
        }  
        if (count > k) {  
            A = A U { nodeWithID(i) }  
        }  
        count = 0  
    }  
    return (max, A)  
}
```

Es.

21. Dato un grafo orientato $G=(V,E)$ ed un albero radicato T con radice g e composto da tutti i nodi V , scrivere un algoritmo in pseudocodice che restituisca TRUE se T corrisponde all'albero DF del grafo G e FALSE altrimenti.

```
DFS(G) {
    A = ∅
    ∀ u ∈ G.V {
        u.color = WHITE
        u.π = NIL
    }
    flag = TRUE
    ∀ u ∈ G.V {
        if u.color == WHITE {
            DFS_VISIT(T, u, flag)
        }
    }
    return flag
}

DFS_VISIT(T, u, flag) {
    u.color = GRAY
    ∀ v ∈ G.adj[u] {
        if v.color == WHITE
            if (v.π != u) flag = FALSE
            DFS_VISIT(T, v, flag)
    }
    u.color = BLACK
}
```