Lezione 12 maggio

Tuesday, May 12, 2020 4:05 PM

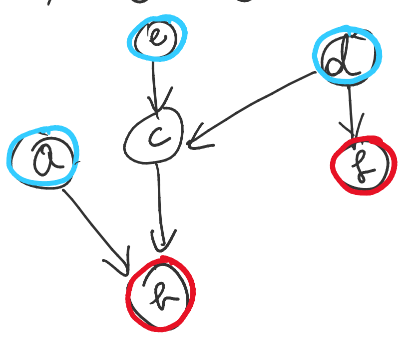
Applicazione della DFS Visita

Classificazione archi - grafi orientati

grafi non orientati < archi dell'albero / archi all'indietro

Ordinamento topologico

input : DAG Directed Acyclic Graph



⇓

vari ordinamenti topologici

source sink

applicazioni

$$\eta : V \to \{0, 1, ..., n-1\}$$

Si parte da un passo z

$$\eta(z) = n-1$$

var. contatore $= n-1$

array : raggiunto[0..n-1] booleano

" : eta [0..n-1] interi

OrdinamentoTopologico(G):

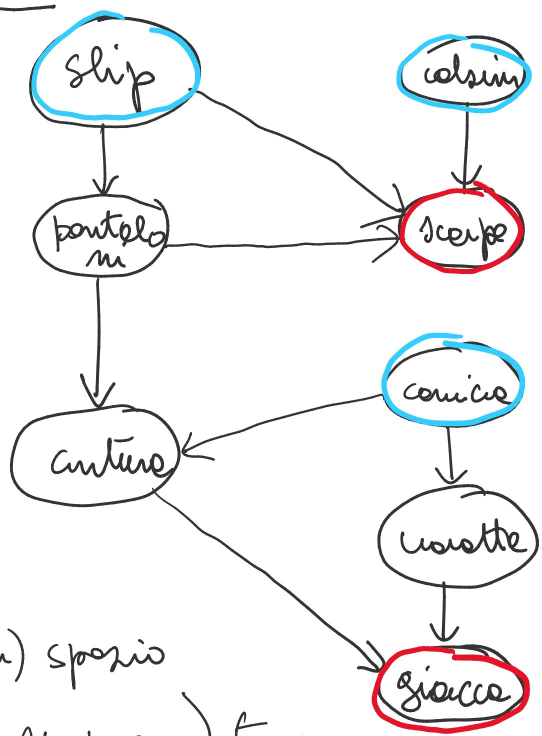
for (s=0; s<n; s++) raggiunto[s]=false;

contatore = n-1

```
for (s = 0; s < n; s++)
    if (!raggiunto[s]) DFS_ordine(s);

DFS_ordine(u):
raggiunto[u] = true;
for (x = Adj[u]; x != NULL; x = x.succ){
    v = x.dato;
    if (!raggiunto[v]) DFS_ordine(v);
} eta[u] = contatore;
  contatore --;
```

DAG



Contatore ~~8~~ ~~7~~ ~~6~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0

DFS_O(slip) 4
DFS_O(pantaloni) 5
DFS_O(cintura) 7
DFS_O(giacca) 8
DFS_O(scarpe) 6
DFS_O(calzini) 3
DFS_O(camicia) 1
DFS_O(cravatta) 2
DFS_O(orologio) 0

$\Theta(n)$ spazio

$\Theta(n + m)$ tempo

Algoritmo di Dijkstra

grafo pesato $G = (V, E, W)$ $W: E \Rightarrow \mathbb{R}$

grafo pesato $G = (V, E, W)$ $W: E \Rightarrow \mathbb{R}$

cammino = $v_0, v_1, \dots, v_k$: $v_i, v_{i+1} \in E$, $0 \le i < k$

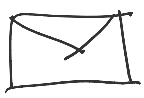
peso del cammino = $\sum_{i=0}^{k-1} W(v_i, v_{i+1})$

<u>input</u>: grafo pesato, sorgente s, $W: E \Rightarrow \mathbb{R}^+$

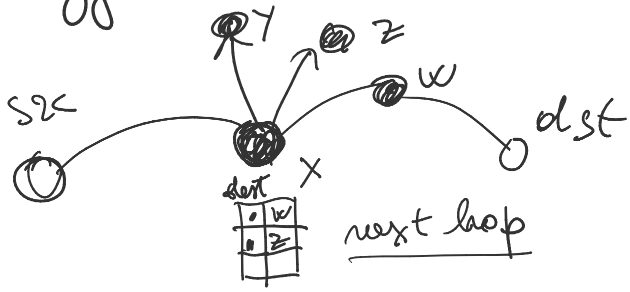
<u>output</u>: il cammino minimo da s a tutti gli altri nodi

<u>shortest path</u>

<u>Routing</u> dei messaggi sulla rete



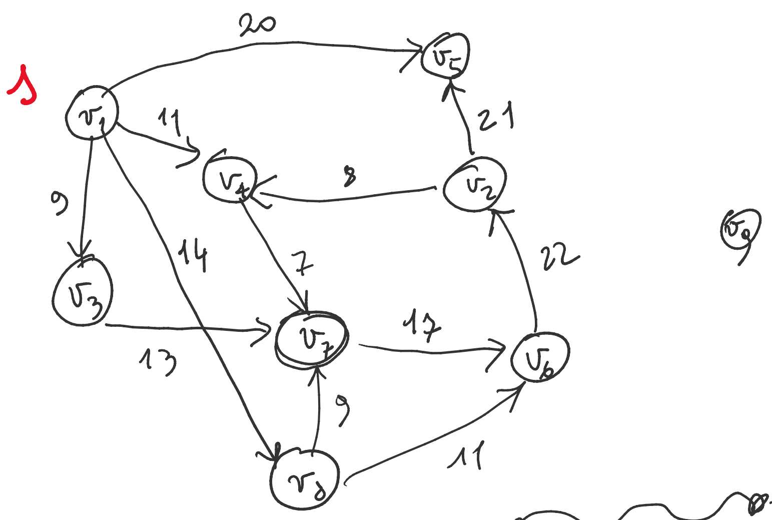
src, dst

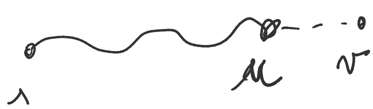


in ogni nodo

tabella di Routing

in accordo allo <u>shortest path</u>





s

$(v_1, 0)$ $(v_2, 57)$ $(v_3, 9)$ $(v_4, 11)$ $(v_5, 20)$ $(v_6, 25)$

$(v_7, 18)$ $(v_8, 14)$ (v_9, ∞) PRED

Usa una coda di priorità (Heap di minimo) ordinata sulle distanze da s.

array PRED[0..n-1] iniz. = -1

array DIST[0..n-1] da s dist[u] = ∞

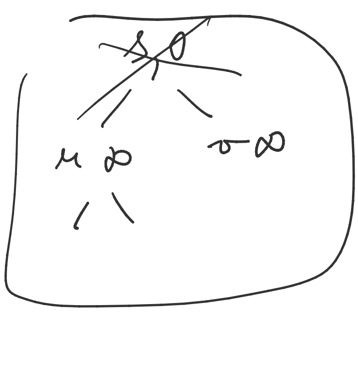
→ nella coda ogni el. è formato da una coppia

elemento.dato = u
elemento.peso = dist[u]

Dijkstra (s):

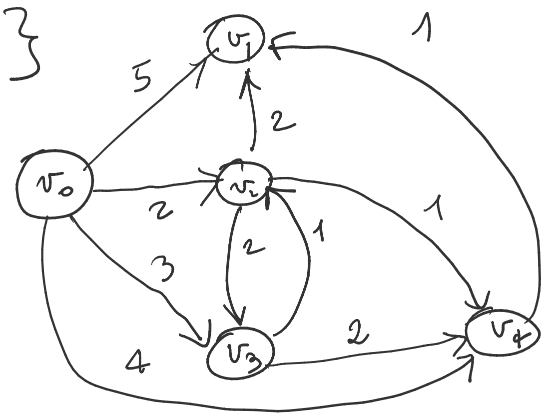
```
for (u=0; u<n; u++) {pred[u]=-1;
      dist[u]=+∞;}
pred[s] = s;
dist[s] = 0;
for (u=0; u<n; u++){
      elemento.peso = dist[u];
      elemento.dato = u;
      Enqueue(PQ, u);
}
while (PQ ≠ ∅) {
```

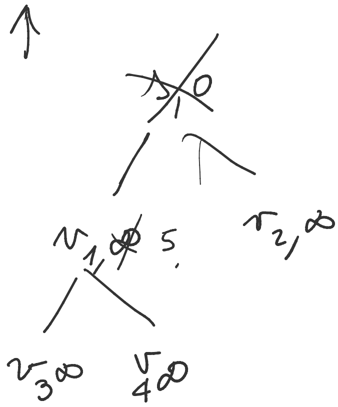
PQ



```
e = Dequeue(P.Q);            s, 0
v = e.dato;
for (x = Adj[v]; x != NULL; x = x.succ){
    u = x.dato;
    if (dist[u] > dist[v] + x.peso){
        dist[u] = dist[v] + x.peso;
        pred[u] = v;
        DecreaseKey(PQ, u, dist[u])
    }
  }
}
```

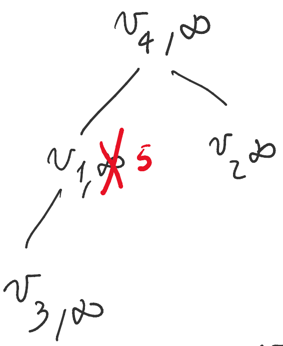
RELAXATION

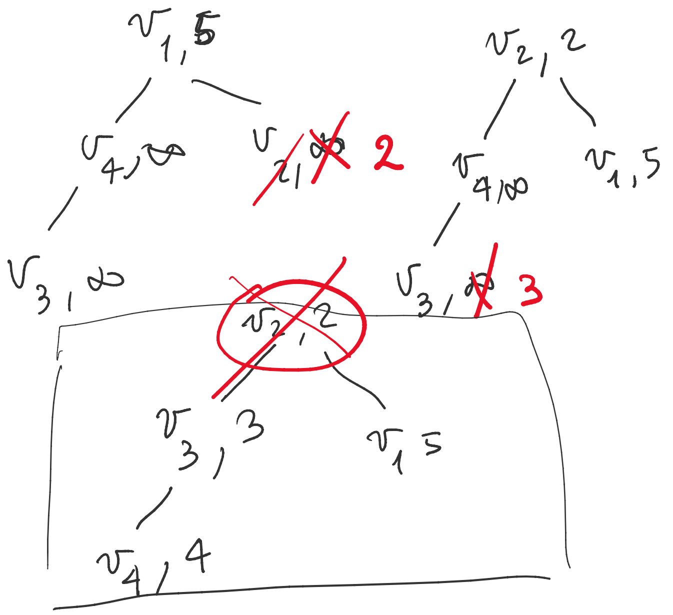


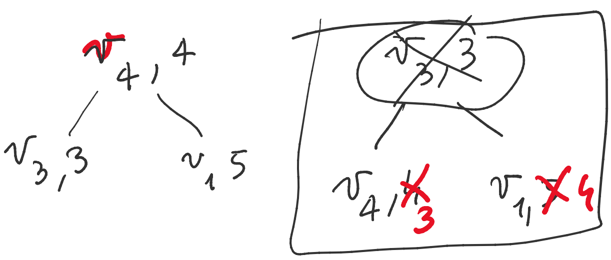


0	1	2	3	4	
0	∅	∅	∅	∅	PRED

0	1	2	3	4	
0	~~∞~~	~~∞~~	~~∞~~	∞	DIST
	~~5~~	2	3	4	



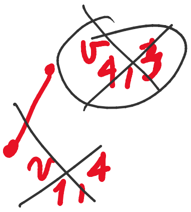




0	1	2	3	4	
0	4	2	3	4	DIST
0	2	0	3	0	PRED

v_2 : v_1, v_4, v_3

5 > 2 + 2



$4 > 2 + 1$

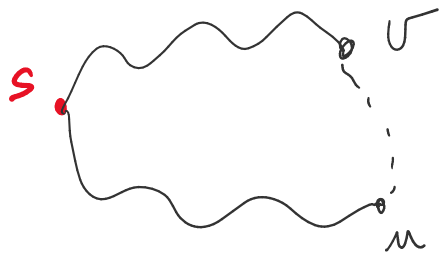
0	1	2	3	4	
0	4	2	3	3	DIST

	1	2	3	4	
∅	2	0	0	2	PRED

$v_3 : v_2, v_4$

$v_4 : v_1$ $\qquad 4 \not> \; = dist[v_4] + v_{4,}v_1$

$v_1 :$



Dequeue $O(\log n)$

DecreaseKey $O(\log n)$ con Riorganizzaheap pg 51

PosHeap $[0 .. n-1]$
mantiene la posizione nell'heap di tutti i nodi.

Alg. Dijstra