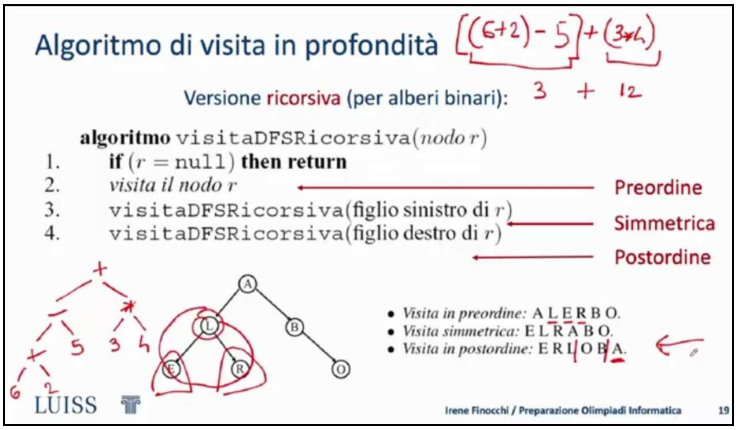
Un **ALBERO** è un grafo non orientato nel quale due vertici qualsiasi sono connessi da uno e un solo cammino (grafo non orientato, connesso e privo di cicli). Per gli **alberi** si possono utilizzare diverse rappresentazioni:

p. 9 per un **albero binario** (al massimo 2 figli, sinistro e destro) per ogni nodo si possono usare 2 puntatori uno al figlio sinistro e l'altro al figlio destro; per un albero non binario ogni nodo punta ad una lista di puntatori a tutti i figli.

p. 11-15 Gli alberi possono essere visitati in modo sistematico (A) https://it.wikipedia.org/wiki/Ricerca_in_profondit%C3%A0

- **visita in profondità (DFS** = Depth First Search) è un algoritmo **ricorsivo** e richiede **uno stack (pila)**

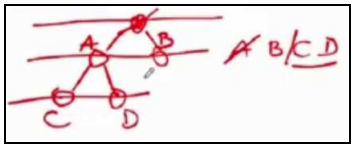


può essere effettuata:

- **in Preordine** (A-p.19-33): prima si visita il **nodo** e poi i suoi **sottoalberi**;
- **Inordine** (se binario) / **Simmetrica** (A-p.34-47): prima si visita il sottoalbero **sinistro**, poi il **nodo** e infine il sottoalbero **destro**;
- **in Postordine** (A-p.48-61): prima si visitano i **sottoalberi**, poi il **nodo**.

← Un esempio di utilizzo delle visite in Postordine: la stampa delle espressioni matematiche.

- **visita in ampiezza (BFS** = Breadth First Search) richiede **una queue (coda)** e si procede **per livelli** →



da sinistra a destra prima il nodo di liv. 0, poi i nodi di liv.1 (A e B) poi nodi di liv. 2 (C e D)

Anche i **GRAFI** possono essere visitati in profondità (DFS) e ampiezza (BFS).

Dato un grafo connesso e con archi non orientati, un **albero ricoprente o di connessione** contiene tutti i vertici del grafo e contiene soltanto un sottoinsieme degli archi, cioè solo quelli necessari per connettere tra loro tutti i vertici con uno e un solo cammino. Per un **grafo con archi pesati** un classico problema è quello della determinazione **dell'albero ricoprente minimo** o **albero di copertura di costo minimo** (**Minimum Spanning Tree**, MST) cioè di un albero ricoprente nel quale, sommando i pesi degli archi, si ottiene un valore minimo.

p. 20 pseudocodice per **algoritmo Visita BFS**

T rappresenterà l'**albero di copertura di radice s** (nodo da cui si intende partire), nodo che va marcato e inserito nella coda **F** (linee 2-5)

Finchè **F** è piena (linee 6-12)

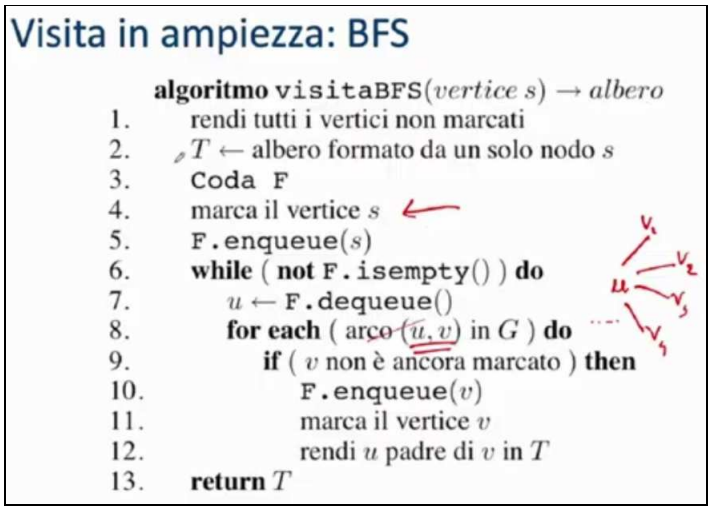
prelevato dalla coda il nodo **u** (=**s** all'inizio)

per ogni arco che collega **u** ai nodi **v**

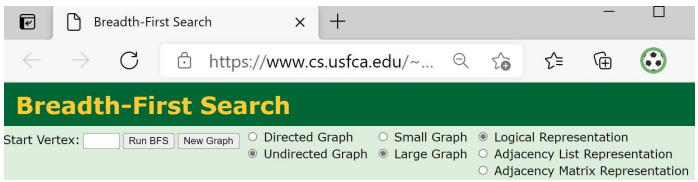
se **v** non è ancora marcato:

- si inserisce nella coda **F** il nodo **v**
- si marca il nodo **v**
- si indica **u** padre di **v** nella struttura **T**

https://www.cs.usfca.edu/~galles/visualization/BFS.htm



La prof.ssa Finocchi utilizzando il link precedente ha mostrato una animazione dell'algoritmo selezionando *Undirected Graph, Large Graph, Logical Representation* e impostando un valore per *Start Vertex* – NOTA è più facile seguire l'andamento dell'algoritmo per *Small Graph*

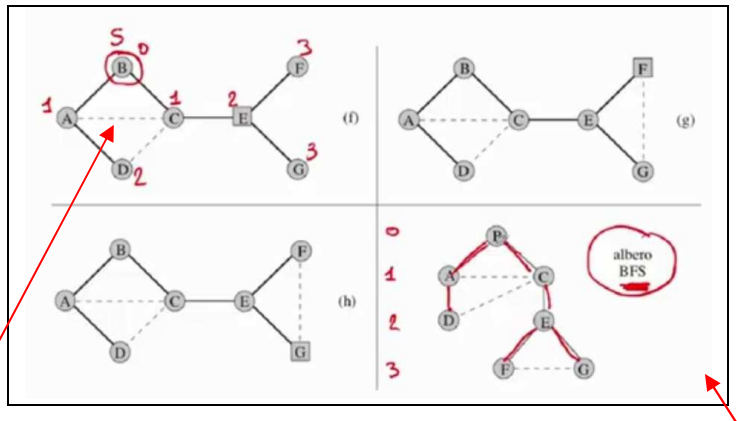


p. 22-23

partendo dal nodo **B**(=***s***), l'algoritmo BFS marca B, A e C ed inserisce nella coda ***F*** il vertice B e i nodi collegati A(=v_1) e C(=v_2) indicandoli in ***T*** come figli di B;

esamina quindi il nodo **A** marcando D ma non C (perchè già marcato partendo da B) nè B nodo di partenza;

viene poi esaminato **C** per il quale va inserito in coda solo il nodo E perchè B, A e D sono già marcati .



NOTA: il percorso da B ad A che passa da C (BCA) è più lungo di quello diretto da B ad A; invece il percorso da B a D che passa da C (BCD) è lungo 2, quindi è equivalente a quello già analizzato (BAD).

pag. 23 L'algoritmo si arresta quando viene raggiunto ed esaminato il nodo **G**. L'ultimo quadrante rappresenta l'**albero di copertura** con radice B del grafo originario; gli archi tratteggiati sono quelli scartati per determinare l'albero di copertura (distanze minime di ogni nodo da B per archi di ugual peso/valore).

p. 25 pseudocodice per l' **algoritmo di Visita DFS**; è **ricorsivo** ed utilizza la pila ***T***; si arresta quando arriva ad elaborare l'ultimo nodo i cui archi raggiungono solo nodi già marcati: nodo **G** nell'esempio pag. 27-28;

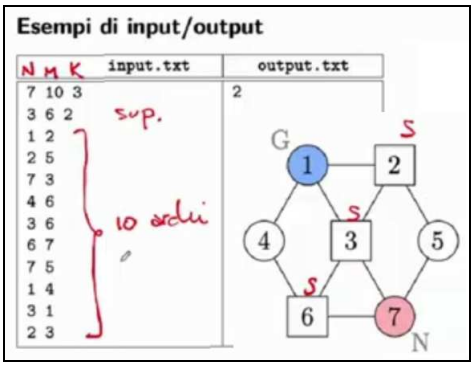
partendo da **B,** si analizza **A**, lo si marca e si invoca nuovamente la funzione *visitaDFSricorsiva (**A**, T)*;

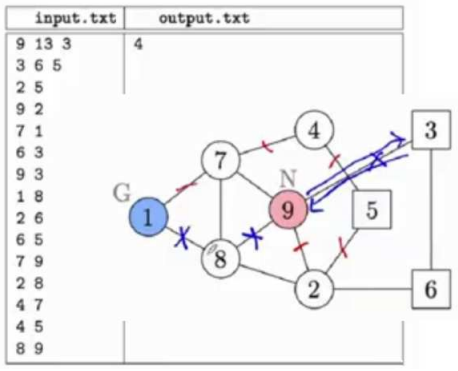
da **A**, saltato B già marcato, si analizza **C**, lo si marca e si invoca nuovamente la funzione *visitaDFSricorsiva (**C**, T);*

da **C,** saltati A e B già marcati, si analizza **D**, lo si marca e si invoca la funzione *visitaDFSricorsiva (**D**, T)* e via così.

programmi C++ per gli algoritmi BFS e DFS a pag 79-81 della *Guida per le selezioni territoriali* del prof.Bugatti

esercizio: **problema Spesa lampo (spesa)**





Il grafo rappresenta gli archi (strade) elencati dalla riga 3 alla riga M+2 del file di input; i supermercati, elencati alla riga 2, sono evidenziati come quadrati.

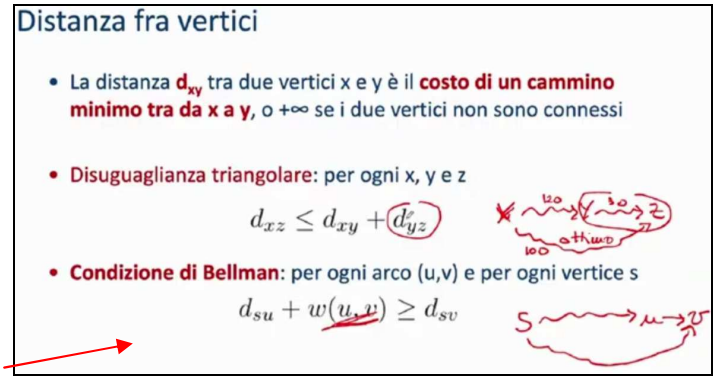
Non è detto che il percorso minimo dalla posizione di Gabriele a casa della nonna passando per un supermercato si ottenga dalla somma delle distanze da un supermercato che minimizza la distanza da Gabriele oppure dalla nonna. Per il secondo caso di test il percorso migliore si ottiene passando da casa della nonna, proseguendo verso il superm. 3 e tornando poi indietro.

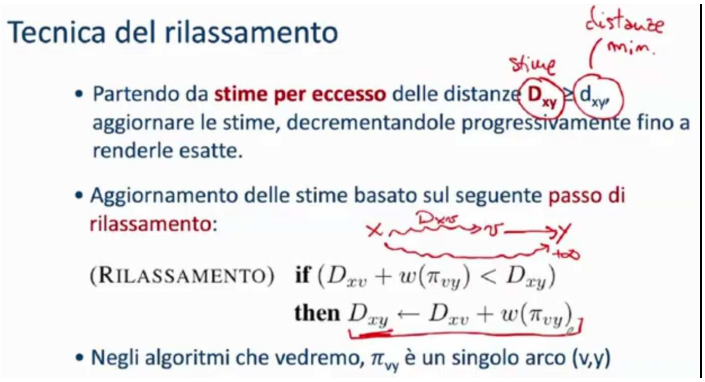
Soluzione proposta da un alunno nella edizione 2020 del corso.

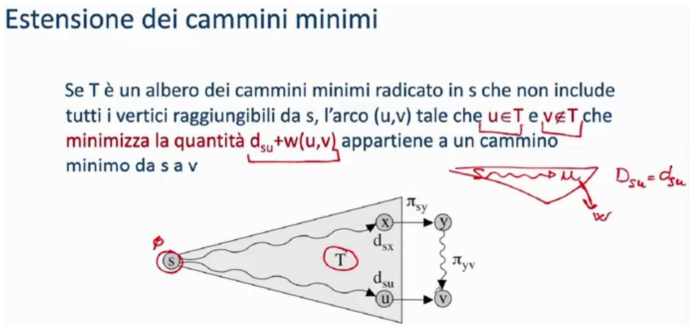
Cammini minimi

$d_{u\,w}$=2+3+10 deve essere ottima altrimenti $d_{u\,v}$ non lo è

w(u,v) è il peso/costo dell'arco **u → v**





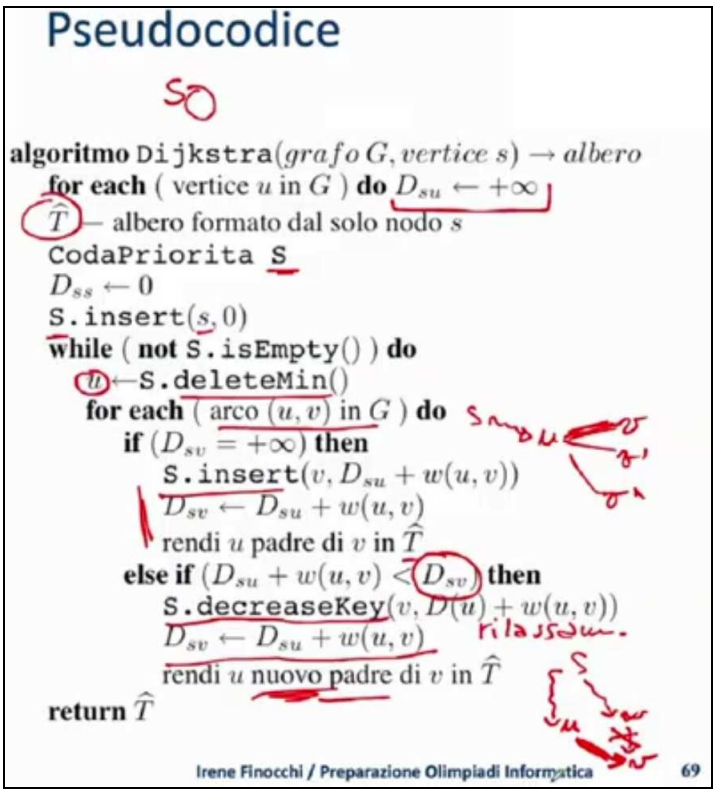


Con d si indicano le distanze minime; con D le distanze stimate ad una certa iterazione.

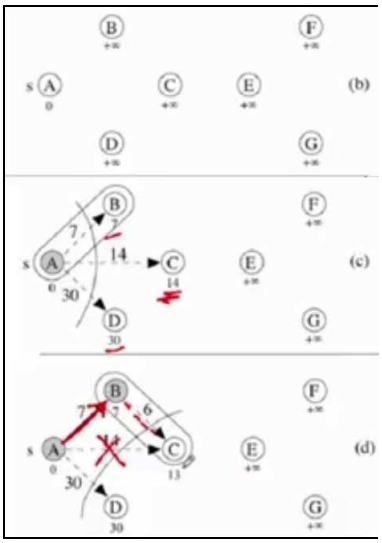
Algoritmo di Dijkstra

Ad ogni passo va trovato un nodo per il quale si possa con certezza dire qual è il cammino minimo da esso verso il nodo di partenza (***s***). Al passo successivo se ne troverà un altro che verrà aggiunto all'insieme dei nodi ***T*** di cui si conosce il cammino minimo e così via fino ad arrivare al nodo di destinazione.

L'algortimo di Dijkstra trova il cammino minimo **da una sorgente verso tutti gli altri nodi del grafo**; si basa sulla **condizione di Bellman** e sulla **Tecnica del Rilassamento**.



Le stime ***D*** vengono inizializzate per eccesso a +infinito; durante l'elaborazione le stime verranno migliorate per la condizione dell'*else if*. Nell'albero ***T*** all'inizio c'è solo la sorgente, poi ci saranno gli altri nodi.



Inizialmente l'albero ***T*** e la coda con priorità ***S*** hanno solo il nodo **A** con D(A)=0 →

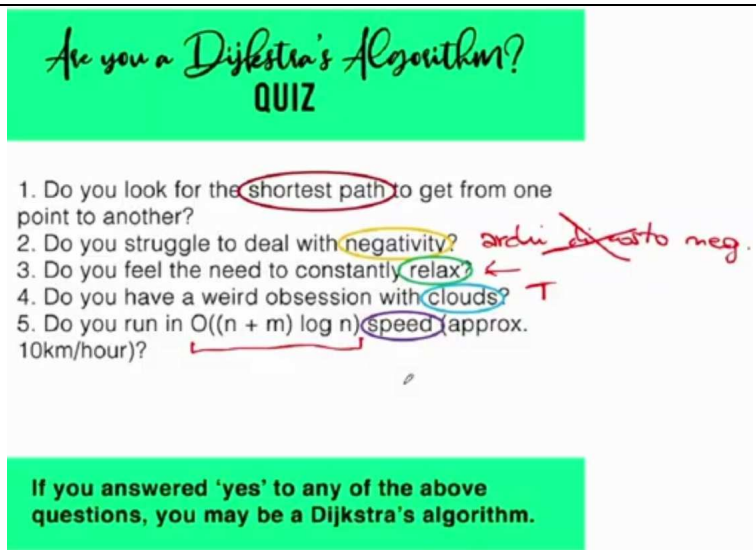
il *while* estrae ed elimina ***u*=A** dalla coda con priorità;

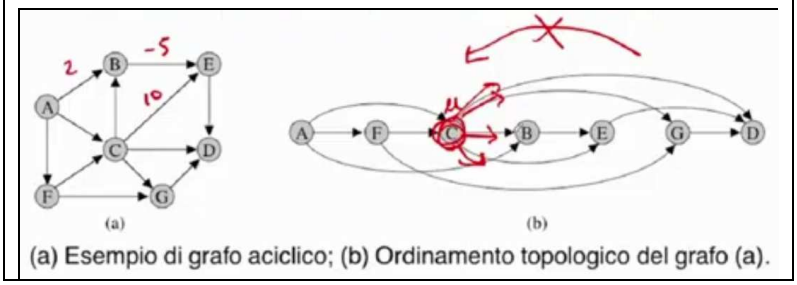
per ***u***=A il *for each* inserisce B C e D nella coda ***S*** e aggiorna i valori D(B)=7, D(C)=14 e D(D)=30, imposta per loro il padre indicando A in ***T***;

il *while* trova di nuovo piena la coda con priorità quindi estrae ed elimina il minimo: ***u*=B** ; il *for each* sul nodo C attiva la *S.decreaseKey,* (condizione *else if* VERA) e quindi migliora la distanza di C e quindi anche la sua posizione nella coda con priorità aggiornandola con il valore D(C) posto a 13=7+6 che risulta migliore del valore precedente, 14.

Il cammino minimo da A a C passa da B: viene pertanto aggiornato l'albero ***T*** impostando come padre di C il nodo B (non più A). Questo è il secondo **Rilassamento** effettuato: ora l'albero ***T*** contiene A, B e C.

ATTENZIONE: la coda con priorità restituisce la priorità massima; per ottenere il minimo vanno memorizzati i valori negativi.





(a) Esempio di grafo aciclico; (b) Ordinamento topologico del grafo (a).

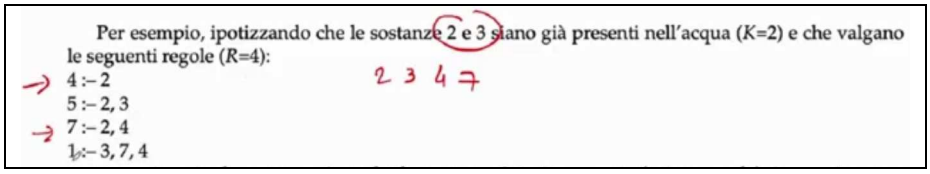
L'algortimo di **Dijkstra** può essere utilizzato **solo per grafi con pesi non negativi**.

Se occorre gestire pesi negativi, per un DAG (Directed Acyclic Graph - grafo orientato senza cicli) c'è un altro algoritmo:

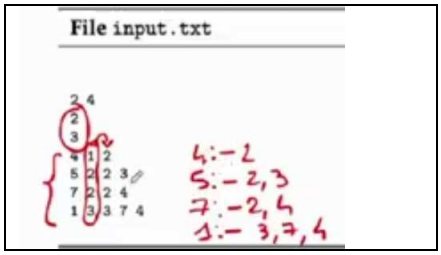
Algoritmo per grafi diretti aciclici

Questo algoritmo esegue i rilassamenti in ordine topologico: in questo ordine si ottengono sempre distanze vere dalla sorgente perchè non ci sarà mai qualche situazione che fa tornare indietro.

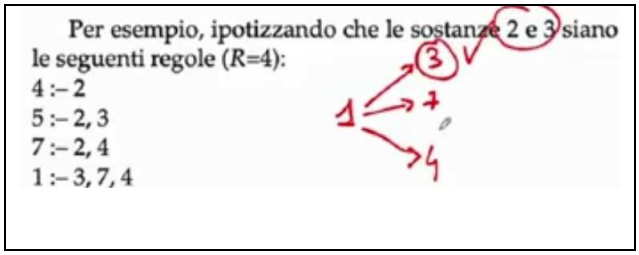
esercizio: problema Depurazione dell'acqua (depura)


Per esempio, ipotizzando che le sostanze 2 e 3 siano già presenti nell'acqua (K=2) e che valgano le seguenti regole (R=4):
4 :– 2
5 :– 2, 3
7 :– 2, 4
1 :– 3, 7, 4

2,3 già presenti si può inserire 4
2 e 4 presenti si può inserire 7
la sostanza 5 non serve perché l'obiettivo è disciogliere la sostanza 1



Si può usare la visita DFS ricorsiva con un vettore delle sosteanze rimanenti partendo dalla 1
ricorsiva sul 3 sul 7 e sul 4


Per esempio, ipotizzando che le sostanze 2 e 3 siano le seguenti regole (R=4):
4 :– 2
5 :– 2, 3
7 :– 2, 4
1 :– 3, 7, 4