Quarto Esonero

corso di laurea in **Matematica**

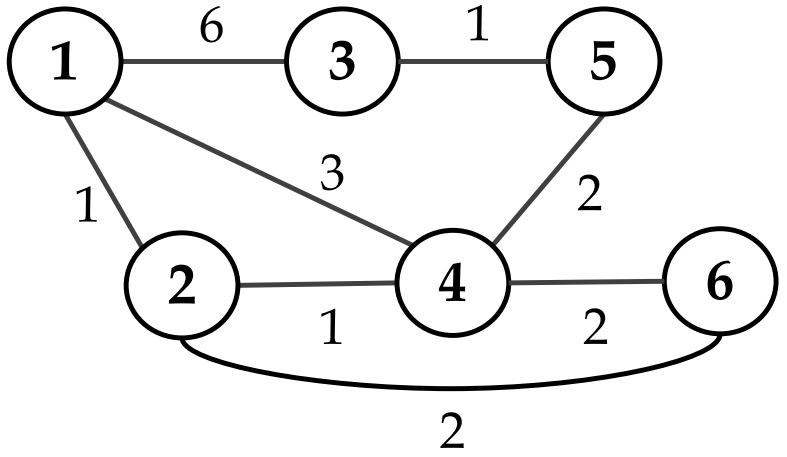
Informatica Generale, Esonero **4** [**7/6/22**]

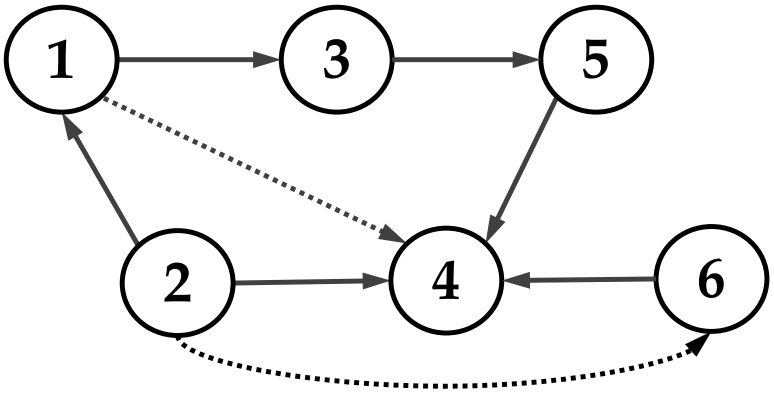
Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Quarto Esonero

Esercizio 1: Del grafo in esame, calcolate (**1**) le distanze in numero di archi dal nodo 1 e (**2**) i cammini di peso minimo dal nodo 1. (**3**) Descrivere un algoritmo per calcolare il *diametro* di un grafo (cioè la più grande distanza tra due nodi) e calcolare il diametro del grafo.





Esempio di orientamento degli archi aggiunti tratteggiati

Esercizio 2: Dato un grafo diretto aciclico $G = (V, E)$ e un insieme di archi non orientati $E' \subseteq V \times V$, descrivere un algoritmo per orientare gli archi in E' in modo che possano essere aggiunti a G senza generare cicli.

Come si possono orientare gli archi di un grafo non orientato per ottenere un DAG?

Soluzioni 4to Esonero: è una questione di Qualità

corso di laurea in **Matematica**

Informatica Generale, Lezione 27 **[10/6/22]**

Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

È una questione di qualità…

I voti sono stati raccolti tra **8** e **10** punti. Il compito era abbastanza congeniato per ottenere questo tipo di risultato.

Metodologìa valutativa: il compito conteneva numerosi quesiti ”semplici”. Ho valutato degne di 1 o 2 punti in più soprattutto alcune finezze, più che la quantità (che cmq è stata tenuta in conto in alcuni casi)

Pro:

- centrato l’obiettivo di aver valutato lo studio e la capacità di concettualizzare semplici problemi;
- chi è arrivato in fondo agli esoneri ha fiutato la possibilità di potercela fare e si è impegnato per ottenere il risultato

Contro:

- alcune finezze sono state ignorate dai più

Nella correzione, mi concentrerò su queste finezze.

Diametro di un grafo

Posto che non ci sono stati particolari problemi a disegnare gli alberi di BFS e a trovare la coppia di nodi più distanti…

L'idea migliore era fare n BFS ciascuna radicata in un diverso nodo del grafo, calcolando le distanze da tutti gli altri nodi.

L'idea migliore però è modificare leggermente **`allDist`** vista a lezione, facendo tornare **solo la distanza massima** (che corrisponde alla distanza dell'ultimo nodo visitato). Scriviamo **`maxDist`**.

```
def maxDist(G, s):
    Q, d = newQueue(), init(numNodi(G), -1)
    d[s], maxd = 0, 0
    enqueue(Q, s)
    while not isEmpty(Q):
        u = dequeue(Q)
        forall v ∈ G.adj(u):
            if d[v] < 0:  enqueue(Q, v)
                          d[v] = d[u] + 1
                          maxd = d[v]
    return maxd
```

$\Theta(n+m)$

```
def diametro(G):
    d = 0
    forall v ∈ V:
        md = maxDist(G, v)
        if md > d: d = md
    return d
```

$\Theta(n(n+m))$

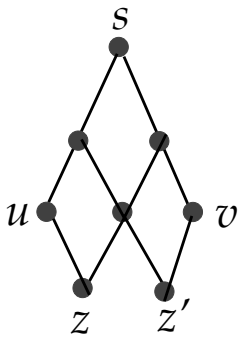
Tentativi anche interessanti

Alcuni hanno pensato di calcolare le distanze tra **tutte le coppie di nodi** e poi **fare il max sulla matrice**.

Tuttavia questa procedura **costa $\Theta(n^3)$ con Floyd-Warshall** solo accennato a lezione [ma potevo essere così cattivo 😇] oppure qualcosa tipo $\Theta(n^3 \log n)$ con prodotto di matrici e poi $\Theta(n^2)$ per calcolare il massimo.

Interessante tentativo chi ha creduto fossero sufficienti **2 sole BFS**, credendo che un nodo che realizza la massima distanza debba essere tra quelli dell'**ultimo livello** di una **qualsiasi BFS**. Questo **è vero per gli alberi**, ma **non per i grafi**.

Ecco un controesempio.



u e v sono **i nodi più distanti**, ma **non sono foglie** dell'albero di visita radicato in s.

Tentativi più discutibili

Ovviamente c'è sempre l'**algoritmo ignorante**, cioè quello che applica la definizione di diametro e quindi calcola $dist(u, v)$ per tutte le coppie di nodi $u, v \in V$. Il costo è $\Theta(n^2(n+m))$.

Un'idea **non ortodossa** è **applicare Dijkstra** usando come pesi degli archi tutti 1: aldilà del costo è chiaro che sto usando una procedura più complicata che risolve un problema più generale.

Per grafi densi è $\Theta(n^3)$, perché comunque va ripetuto n volte, non molto diverso da $\Theta(n^2 + nm)$ che si ottiene con la BFS.

Per grafi sparsi è $\Theta(n^2 \log n)$, leggermente superiore a $\Theta(n^2)$ che si ottiene con la BFS.

Orientamento di archi

Siccome G **è un DAG**, ammette un **ordine topologico** $\leq_{TS}$, e quindi oriento tutti gli archi di E' in accordo con $\leq_{TS}$. L'idea è corretta perché $\leq_{TS}$ è un **ordine topologico** anche **per** G^*, che quindi è un DAG.

La finezza era: **come calcolare** $\leq_{TS}$ **in** $\theta(1)$**?** Ovviamente la lista ordinata nell'ordine topologico aiutava poco a questo scopo…

Pensando all'algoritmo che iterativamente sceglie nodi con grado entrante 0, si può costruire un vettore **TS** indicizzato sui nodi che registra il **tempo di scelta** di un nodo. Quindi $u \leq_{TS} v$ **sse** $TS[u] < TS[v]$.

Pensando all'algoritmo che ordina in senso inverso ai tempi di uscita di una DFS… beh, **bastano i tempi di uscita**! In quel caso $u \leq_{TS} v$ **se e solo se** $out[u] > out[v]$.

Chiamando $p = |E'|$ il tutto costa $\theta(n+m+p)$.

$\theta(n+m)$

```
def orienta(G, E'):
    ≤TS = topologicalSort(G)
    forall (u, v) ∈ E':
        if u ≤TS v: E = E ∪ { u → v }
        else: E = E ∪ { v → u }
    return G
```

$\theta(p)$

Piccole finezze

Quando è possibile orientare un arco **in entrambe le direzioni**?

Quando **non c'è** un **cammino** da u **a** v, né **da** v **a** u. Detto in altri termini, quando c'è sia un ordinamento topologico $\leq_T$ tale che $u \leq_T v$, sia un ordinamento topologico $\leq_{T'}$ tale che $v \leq_{T'} u$.

Infine: **come orientare un generico grafo non orientato** in modo da non creare cicli? Una **visita, certo** (sia BFS che DFS, ma in entrambi i casi occorre **fare attenzione agli archi non dell'albero** di visita).

Però **è sufficiente scegliere un ordinamento topologico arbitrario**, ad esempio quello indotto dalla codifica dei nodi con numeri interi!

```
def orienta(G):
    forall (u, v) ∈ E':
        if u ≤ v:  E = E ∪ { u → v }
        else: E = E ∪ { v → u }
    return G
```

$\theta(m)$