# SCC
# n vertici and probability

Vivoli Emanuele

Luglio 2017

# 1 Introduzione

L'esercizio consiste nello studio dell'algoritmo Strongly-Connected-Components (ricerca delle componenti fortemente connesse). In particolare si richiede l'uso dell'algoritmo Depth-First-Search che permette una visita del grafo in profondità.

# 2 La Teoria

Vediamo gli aspetti teorici del problema, in particolare i 2 algoritmi presentati nel paragrafo precedente, le aspettative che si hanno sui risultati e una descrizione più approfondita dell'esperimento.

## 2.1 DFS

La Depth-First-Search è una visita in profondità applicata ai grafi. Ad ogni nodo vengono aggiunti degli attributi utili alla visita:

1. *color* : indica lo stato del nodo :
   (a) $WHITE$ : nodo non ancora scoperto
   (b) $GRAY$ : nodo scoperto, ma non terminato
   (c) $BLACK$ : nodo terminato
2. $d$ : tempo di scoperta (quando un nodo viene scoperto per la prima volta durante la visita)
3. $f$ : tempo di terminazione (quando il nodo diventa $BLACK$)
4. $\pi$ : nodo precedente nella visita

Si hanno delle limitazioni su $d$ ed $f$, infatti vale che $1 \leq d < f \leq 2|V|$ , dove $|V|$ è il numero di nodi presenti nel grafo.

Alla fine dell'algoritmo ogni nodo avrà il suo tempo di scoperta, quello di terminazione e il nodo precedente ($\pi$). In base a questi riesco a costruire molteplici sotto-grafi che vanno a formare una foresta DF, contenente quindi almeno un albero DF. La complessità è pari a $\Theta(V + E)$ perchè l'algoritmo esplora sicuramente tutti i nodi e tutti gli archi.

## 2.2 SCC

Dato il grafo G=(V,E) una componente fortemente connessa, detta anche Strongly Connected Component (SCC), è un insieme massimale di vertici $C \subseteq V$ tale che per ogni $u, v \in C$ esistono entrambi i cammini $u \hookrightarrow v$ e $v \hookrightarrow u$. Per trovare queste SCC viene usato un algoritmo che a sua volta utilizza la visita DFS, l'algoritmo infatti è composto da questi 4 passaggi:

- applicare DFS sul grafo G (calcolando quindi per ogni nodo $f$)
- calcolare il trasposto di G (stesso numero di nodi ma archi invertiti)
- applicare DFS sul grafo $G^T$ visitando i nodi in ordine decrescente rispetto all'attributo $f$
- generare la foresta DF prodotta dal DFS del punto precedente

# 3 Aspettative

Prima di studiare i risultati pratici che l'esecuzione del programma ci potrà mostrare, possiamo già fare un'analisi riguardo al comportamento che ci aspettiamo.

Ci aspettiamo che l'algoritmo SCC impieghi più tempo con l'aumentare del numero di nodi e archi. Inoltre ci aspettiamo che all'aumentare del numero di archi ci siano sempre meno alberi DF, mentre con lo stesso numero di nodi ma numero di archi minore ci aspettiamo un numero più alto di alberi DF.

# 4 Descrizione esperimenti

L'esercizio propone di creare grafi casuali con numero di nodi n e probabilità di avere un arco tra 2 nodi prob; si chiede inoltre di fare dei test con grafi crescenti sia di dimensioni (e quindi con l'aumento di n) sia di archi (e quindi con l'aumento di prob). E' stato deciso allora di fare dei test con grafi di dimensioni da 0 a 50 di passo 5. Per ogni dimensione inoltre vengono analizzati i casi dove la probabilità di presenza di archi è uguale a '15%', '30%', '45%', '60%', '75%' (la probabilità potrebbe comunque restare inferiore rispetto ai valori massimi assegnati per i test, perchè già con 50% si trova, quasi sempre, una sola componente fortemente connessa). Il test calcola il tempo impiegato per trovare le SCC, il numero di alberi DF prodotti e la media dei nodi presenti negli alberi DF (con il numero di alberi DF che aumenta, si avrà una diminuzione del numero di nodi per albero).

## 4.1 Codice di calcolo

Codice relativo alla struttura degli oggetti Vertice e Grafo che vengono implementati:

```
# Vertice
class V:
    def __init__(self, key):
        self.key = key
        self.color = None
        self.p = None
        self.d = None
        self.f = None


# Graph
class G:

    def __init__(self, n, prob):
        self.SCC = []
        self.V = []
        self.time = 0

        # create a group of Summit
        for i in range(n):
            self.V.append(V(i))

        # create matrix |n| x |n| of 0
        self.E = [[0 for i in range(n)]
                        for j in range(n)]
        for i in range(n):
            for j in range(n):
                    # set   1 with probability 'prob'
                    #       0 with probability '1-prob'
                    if random.randint(1, 100) <= prob:
                        self.E[i][j] = 1
                    else:
                        self.E[i][j] = 0
```

Codice relativo ai test che vengono effettuati per il calcolo di SCC:

```
n = range(0, 50, 2)
prob = range(15, 90, 15)

t = [[] for i in n]
num = [[] for i in n]
ln = [[] for i in n]

for i in range(len(n)):

    t[i] = [0 for j in range(len(prob))]
    num[i] = [0 for j in range(len(prob))]
    ln[i] = [0 for j in range(len(prob))]

    for j in range(len(prob)):
        for k in range(20):

            g = G(n[i], prob[j])
            start = time()
            SCC(g)
            end = time()

            len_m = 0
            for l in range(len(g.SCC)):
                len_m += len(g.SCC[l])
            len_m /= (1 if len(g.SCC) == 0
                           else len(g.SCC))

            t[i][j] = (t[i][j] * k +
                       (end - start)) / (k + 1)
            num[i][j] = (num[i][j] * k +
                         len(g.SCC)) / (k + 1)
            ln[i][j] = (ln[i][j] * k +
                                len_m) / (k + 1)
```

# 5 Documentazione

Per l'esercizio sono state implementati 3 file : "Graph.py", "test.py" e "exp.py"; nei seguenti paragrafi ne vediamo i dettagli.

## 5.1 Graph.py

In questo file sono state implementati le classi principali e delle funzioni utili. Segue un elenco dettagliato degli elementi all'interno del file:

- classe V

  La classe rappresenta un nodo (o vertice) nel grafo. Ha come attributi *key* (per identificarlo), *color* (usato all'interno degli algoritmi per sapere se è stato visitato o meno), *pi* (usato per ricondursi alla foresta DF), $d$ (il tempo di scoperta), $f$ (il tempo di terminazione).

- classe G

  La classe rappresenta un grafo costruito a partire da una matrice di incidenza o da numero di elementi e probabilità di presenza archi. Ha come attributi $V$ (la lista dei vertici presenti nel grafo), $E$ (che rappresenta la matrice di incidenza del grafo). Di vitale importanza per l'esercizio è il metodo che trova le componenti fortemente connesse; esso usa la funzione di visita DFS e la funzione che rende il grafo trasposto (con gli archi inversi), come visto a lezione.

## 5.2 test.py

All'interno di questo file è stato implementato il test, come si puo ben capire dal nome, ed esegue per ogni valore di n (numero di vertici), e per ogni valore di proabilità impostato (15,30, 45, 60, 75) la funzione SCC 20 volte calcolando sul numero totale di iterazioni la media dei tempi d'esecuzione, del numero di alberi DF e dei nodi presenti all'interno degli alberi DF. il risultato viene salvato infine in file pickle, ed utilizzato dal programma exp.py per la visualizzazione dei risultati sperimentali.

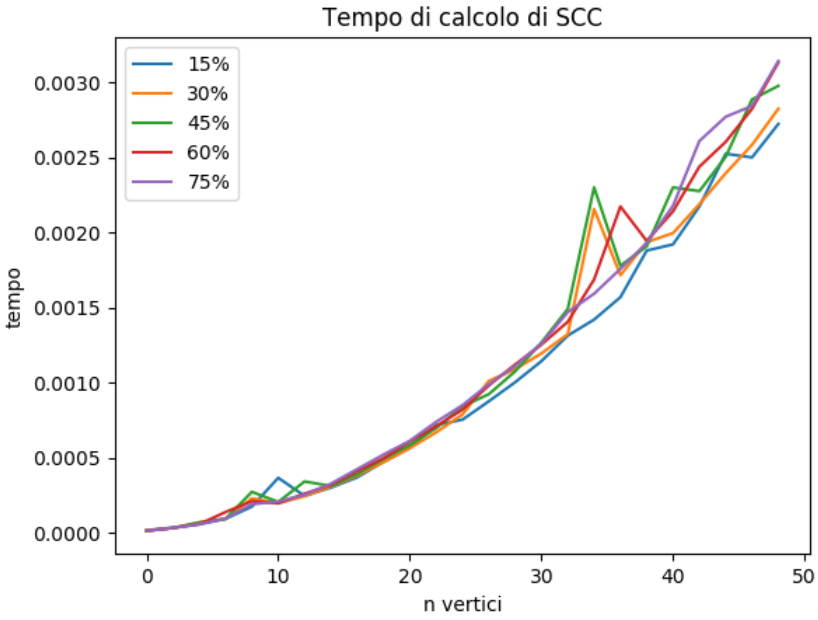
## 5.3 exp.py

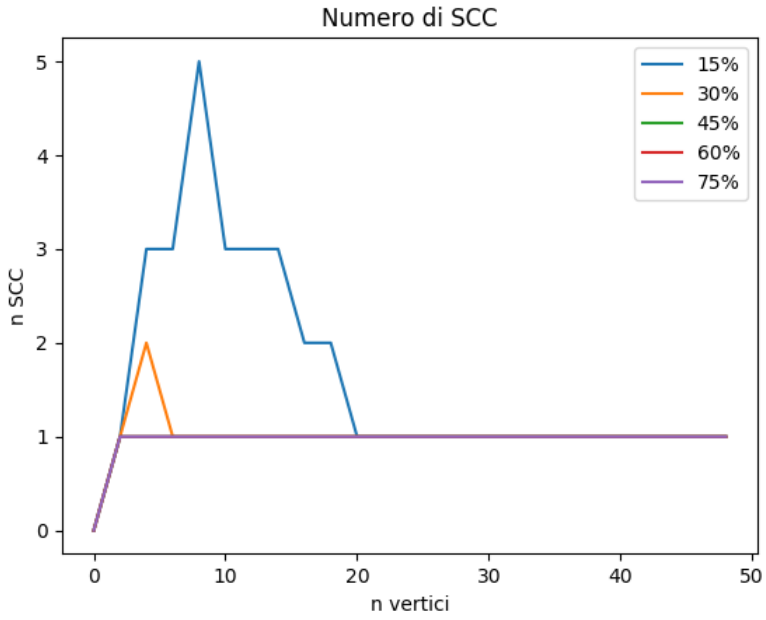
All'interno di quest'ultimo file è implementato il codice per la costruzione dei grafici in base ai risultati dei test. Come risultato avremo 3 grafici , uno per il tempo impiegato, uno per il numero di alberi DF ed uno per il numero medio di nodi appartenenti agli alberi. Ogni grafico poi avrà x funzioni con x pari al valore della probabilità utilizzata nei test (nel nostro caso x=5 (15%, 30%, 45%, 60%, 75%).

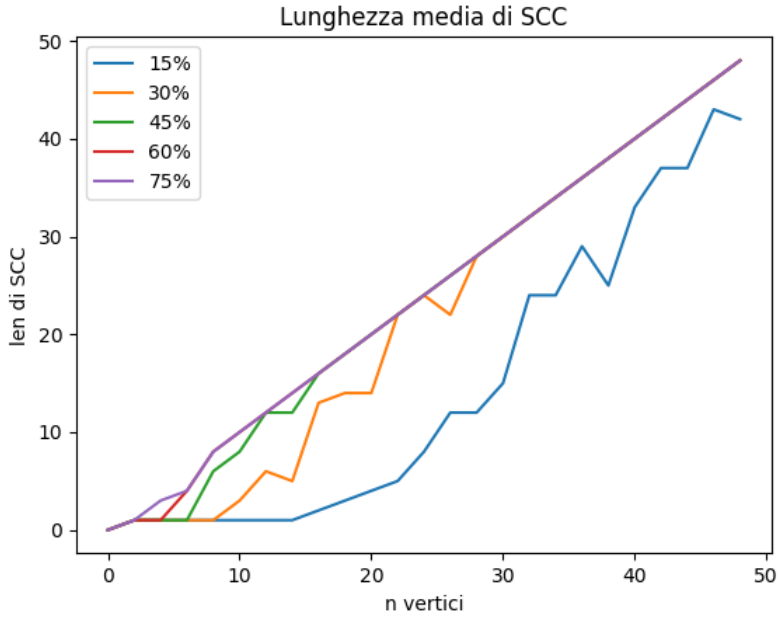
# 6 Risultati Sprimentali

I risultati sono quelli che ci aspettavamo: il tempo impiegato aumenta con l'aumentare della dimensione del grafo; il numero di alberi va a diminuire con l'aumentare della probabilità di presenza archi; il numero medio di nodi invece va ad aumentare (avrà con probabilità 1 solo un albero DF contenente tutti i nodi del grafo).

Dalla seguente pagina in poi sono riportarti i grafici relativi ai test. Tali grafici ribadiscono i risultati ottenuti ed attesi.




Numero di SCC
n SCC
n vertici
15%
30%
45%
60%
75%
0
1
2
3
4
5
0
10
20
30
40
50


Lunghezza media di SCC
len di SCC
n vertici
15%
30%
45%
60%
75%
0
10
20
30
40
50
0
10
20
30
40
50