ASD2 Homework 2

Valerio Pro

Gennaio 2022

Esercizio 1

Per questo problema ho definito un algoritmo la cui analisi si basa sul *metodo del valore atteso*. Il criterio è:

Scegli u.a.r. un sottoinsieme W di k nodi finchè il numero di archi del sottografo $G[W]$ è strettamente minore della quantità target

Di seguito lo pseudocodice:

Algorithm 1

Require: Un grafo $G = (V, E)$ e un intero $k \in \mathbb{N}$ *con* $k \leq n$
Ensure: Un sottoinsieme di nodi $W \subseteq V$ tale che il grafo $G[W]$ indotto da W contiene almeno $\frac{mk(k-1)}{n(n-1)}$ archi

```
W ← ∅
while |F| < mk(k−1)/n(n−1) do
    Scegli u.a.r. W ⊆ V con |W| = k
end while
return W
```

L'obiettivo è determinare il numero atteso di iterazioni eseguite, trovando una stima della probabilità di successo all'interno del ciclo *while*. Se abbiamo a disposizione tale probabilità p allora il numero atteso di iterazioni sarà semplicemente $\frac{1}{p}$. Definiamo la *v.a.* X:

$$X := \textit{numero di archi in } F \textit{ alla fine della generica iterazione}$$

Prima di tutto verrà dimostrato che il valore atteso di X è esattamente il minimo numero di archi che vogliamo in F, cioè $\frac{mk(k-1)}{n(n-1)}$.

Possiamo scrivere X come somma di m variabili aleatorie booleane, $X = \sum_{(u,v)\in E} X_{(u,v)}$, dove:

$$X_{(u,v)} = \begin{cases} 1, \ \textit{se } (u,v) \in F \\ 0, \ \textit{altrimenti} \end{cases}$$

Adesso che abbiamo introdotto queste m variabili aleatorie booleane, diventa più semplice definire $E[X]$, infatti, usando la linearità del valore atteso:

$$E[X] = E[\sum_{(u,v)\in E} X_{(u,v)}] = \sum_{(u,v)\in E} E[X_{(u,v)}]$$

Dalla definizione di valore atteso segue che:

$$E[X_{(u,v)}] = P[X_{(u,v)} = 1]$$

Cerchiamo di capire quanto vale questa probabilità. Scelto W u.a.r. e fissato un arco $(u, v) \in E$, ci chiediamo quale è la probabilità che questo arco appartiene a F, ovvero quale è la probabilità che entrambi i vertici u e v si trovano in W. Possiamo descrivere questa situazione attraverso uno schema $\frac{Casi\ Favorevoli}{Casi\ Totali}$, in cui:

$Casi\ Favorevoli\ =\ numero\ di\ sottoinsiemi\ W\ che\ contengono\ sia\ u\ che\ v$
$Casi\ Totali\ =\ numero\ di\ sottoinsiemi\ W$

Banalmente i casi totali sono tutti i possibili sottoinsiemi di k nodi presi da un insieme di n nodi, quindi $Casi\ Totali\ =\ \binom{n}{k}$. Per i casi favorevoli immaginiamo di avere i due nodi u, v fissati all'interno di W, per ottenere tutti i sottoinsiemi favorevoli possiamo far variare i restanti $k-2$ nodi (che completano W) presi da un insieme di $n-2$ nodi rimanenti. I sottoinsiemi di k nodi favorevoli per la coppia u, v sono quindi $\binom{n-2}{k-2}$.

Otteniamo che:

$$P[X_{(u,v)} = 1] = \frac{\binom{n-2}{k-2}}{\binom{n}{k}} = \frac{k!(n-2)!}{n!(k-2)!} = \frac{k(k-1)}{n(n-1)}$$

Dal calcolo del valore atteso di X che è una somma di m di queste probabilità:

$$E[X] = \sum_{(u,v)\in E} P[X_{(u,v)} = 1] = \frac{mk(k-1)}{n(n-1)}$$

Poichè $E[X] = \frac{mk(k-1)}{n(n-1)}$ deve necessariamente risultare:

$$\begin{cases} P[X \geq \frac{mk(k-1)}{n(n-1)}] > 0 \\ P[X \leq \frac{mk(k-1)}{n(n-1)}] > 0 \end{cases}$$

Chiaramente siamo interessati alla prima probabilità $p = P[X \geq \frac{mk(k-1)}{n(n-1)}]$, infatti vogliamo darne un lower bound al fine di inquadrare il numero atteso di iterazioni necessarie per trovare un sottoinsieme W target. Chiamiamo $\mu = \frac{mk(k-1)}{n(n-1)}$ e con il metodo del valore atteso proviamo a trovare un lower bound a p.

$$\mu = E[X] = \sum_{k=0}^{m} kP[X=k] = \sum_{k<\mu} kP[X=k] + \sum_{k\geq\mu} kP[X=k] \leq$$
$$\leq \sum_{k<\mu} kP[X=k] + m\sum_{k\geq\mu} P[X=k] \leq$$
$$\leq (\mu-1)\sum_{k<\mu} P[X=k] + m\sum_{k\geq\mu} P[X=k] =$$
$$= (\mu-1)(1-p) + mp =$$
$$= \mu - \mu p - 1 + p + mp$$

Dalla disuguaglianza $\mu \leq \mu - 1 - \mu p + p + mp$, ricaviamo:

$$p \geq \frac{1}{1+m-\mu} = \Theta(\frac{1}{m-\mu})$$

Se p è la probabilità di successo in una generica iterazione allora vengono eseguite $\Theta(m-\mu)$ iterazioni in valore atteso. Il controllo nel *while* può essere eseguito in tempo $\mathcal{O}(nm)$ (bound abbastanza lasco) e anche il campionamento può essere eseguito in tempo polinomiale nella size di G. Sapendo che $k \leq n$, il *Running Time atteso* è polinomiale ed è:

$$Time(n,m,k) = \Theta((m-\mu)poly(n,m)) = \Theta((m - \frac{mk(k-1)}{n(n-1)})poly(n,m))$$

Visto che $\frac{k(k-1)}{n(n-1)} \leq 1$, abbiamo *worst case* ($k$ piccolo rispetto n):

$$Time(n,m) = \mathcal{O}((m)poly(n,m))$$

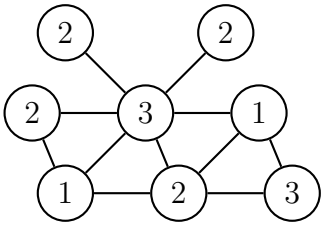
Esercizio 2

2.1

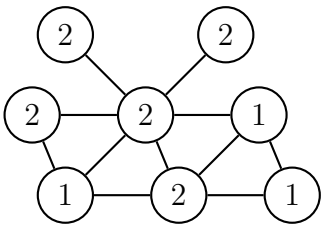
Preso un grafo $G = (V, E)$ sappiamo per definizione che questo grafo è 3-colorabile se esiste un 3-labeling $f : V \to \{1, 2, 3\}$ tale che $f(u) \neq f(v)$ $\forall (u, v) \in E$. L'esistenza della 3-colorazione implica che l'insieme dei nodi V è partizionabile in 3 sottoinsiemi V_1, V_2, V_3 dove $V_i = \{v \in V : f(v) = i\}$ $\forall i = 1, 2, 3$, in particolare, ciascun insieme V_i deve essere necessariamente un *Independent Set*.

Osserviamo quindi che, data una 3-colorazione per G, i vertici di un generico triangolo devono trovarsi separati nelle 3 partizioni di V, prendendo allora i vertici in V_3 e spostandoli arbitrariamente in V_1 o in V_2 (cioè $f(v) = 1$ o $f(v) = 2$ $\forall v \in V_3$), otteniamo un 2-labeling di G in cui nessun triangolo può essere monocromatico. Infatti, dato che V_3 era un Independent Set, nessun vertice con etichetta 3 era adiacente ad altri vertici etichettati allo stesso modo, ciò significa che ogni triangolo avrà adesso due vertici appartenenti a V_1 e un vertice in V_2 oppure un vertice appartenente a V_1 e due vertici in V_2.

Per esempio, dato il grafo seguente con label che forniscono una 3-colorazione:



Otteniamo il 2-labeling:



2.2

Per questo problema ho voluto provare ad ottenere complessità polinomiale e ho definito un algoritmo il cui funzionamento e analisi si basano sulle *Catene di Markov*, di seguito lo pseudocodice

Algorithm 2

Require: Un grafo $G = (V, E)$

Ensure: Un 2-labeling di V tale che, se G è 3-colorabile, non contiene triangoli monocromatici

```
f ← 2-labeling di V arbitrario

for t volte do
    if non ci sono triangoli monocromatici then return f
    end if
    T ← triangolo monocromatico arbitrario
    Scegli vertice v in T u.a.r.
    Cambia l'etichetta di v
end for

return f
```

Prima di entrare nell'analisi formale dell'algoritmo vorrei fornire l'insieme di fatti da cui sono partito e l'intuizione che si basa sul paragrafo precedente:

- se il grafo G di input non ammette il 2-labeling che cerchiamo, indipendentemente dal numero di volte che eseguiamo il *for*, l'algoritmo restituisce un 2-labeling senza la proprietà cercata. Se l'algoritmo ritorna un 2-labeling f allora quel labeling ha la proprietà che cerchiamo[1];
- l'algoritmo sbaglia quando G è 3-colorabile ed esiste un 2-labeling che può essere ritornato ma all'interno del *for* non si riesce a costruire tale labeling. Vogliamo sapere se è possibile scegliere t sufficientemente grande da rendere l'algoritmo corretto *w.h.p.*, mantenendo però una complessità polinomiale nella size di G;
- analizzare il caso in cui il grafo G è 3-colorabile ci fornisce un metro di paragone tra il 2-labeling che stiamo costruendo e la 3-colorazione di G, per questo d'ora in poi si assume che il grafo sia 3-colorabile.

[1]Tuttavia, in questo caso, G potrebbe essere anche non 3-colorabile. Si pensi ad esempio al grafo completo con 4 nodi che non è 3-colorabile ma ammette un 2-labeling in modo da non avere triangoli monocromatici

Sia quindi il grafo G di input 3-colorabile e sia $g : V \to \{1, 2, 3\}$ una sua 3-colorazione, sappiamo che questa 3-colorazione induce una 3-partizione di V in V_1^g, V_2^g, V_3^g.

Sia invece $f : V \to \{1, 2\}$ il 2-labeling costruito dall'algoritmo.

Ad alto livello vogliamo capire quanto tempo impiega il 2-labeling f a etichettare i nodi delle partizioni V_1^g, V_2^g (indotte dalla 3-colorazione) con le rispettive etichette 1 e 2. In altre parole si vuole determinare l'istante in cui ogni nodo in V_1^g è etichettato da f con 1 e ogni nodo in V_2^g è etichettato da f con 2. Se ci troviamo in una situazione del genere, indipendentemente da come sono etichettati da f i nodi in V_3^g avremo che ogni triangolo non è monocromatico (per quanto già dimostrato nel paragrafo 2.1) e l'algoritmo ritorna quindi il 2-labeling desiderato.

Il *Running Time* dipende dal valore che si sceglie per t e dall'enumerazione e controllo dei triangoli all'interno del *for*. Tale enumerazione richiede $\mathcal{O}(n^3)$ poichè possiamo avere al massimo $\binom{n}{3}$ triangoli. Il valore di t verrà determinato nell'analisi formale, per ora diciamo che il *Running Time* è $\mathcal{O}(n^3 t)$.

2.3

Provo di seguito ad analizzare rigorosamente l'algoritmo, ricordando che siamo sotto l'ipotesi che il grafo G di input è effettivamente 3-colorabile. Definiamo innanzitutto cosa stiamo misurando attraverso la successione di variabili aleatorie $\{X_i\}_{i=1,\dots,t}$, dove:

$X_i :=$ numero di nodi in V_1^g e V_2^g su cui f è in accordo con il 3-coloring g all'i-esima iterazione del *for*

Definiamo $k := |V_1^g| + |V_2^g|$ e la $v.a.$ T come:

$$T := \ min\{i : X_i = k\}$$

Nel momento in cui scegliamo un triangolo monocromatico possono accadere tre cose, supponiamo che le etichette del triangolo siano tutte 1:

1) viene cambiata l'etichetta di un nodo v e ci avviciniamo allo stato target, cioè $f(v) = 2$ e $v \in V_2^g$;

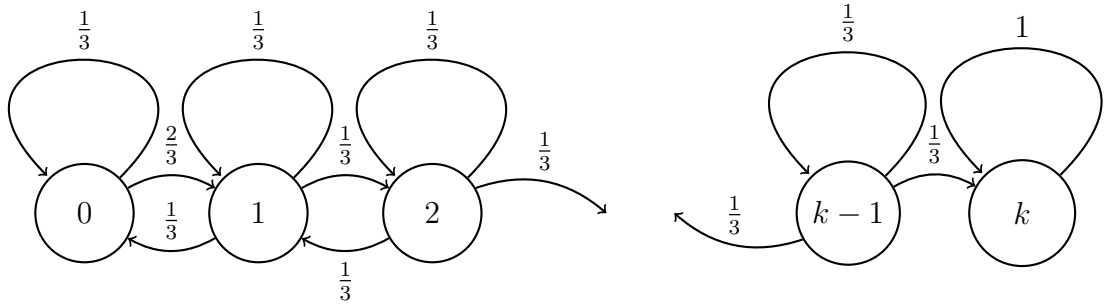
2) viene cambiata l'etichetta di un nodo v e ci allontaniamo dallo stato target, cioè $f(v) = 2$ e $v \in V_1^g$;

3) viene cambiata l'etichetta di un nodo v e rimaniamo alla stessa distanza dallo stato target, cioè $f(v) = 2$ e $v \in V_3^g$.

Dato che la scelta del nodo nel triangolo monocromatico avviene *u.a.r.* e sapendo che la 3-colorazione g assegna etichette diverse ad ogni triangolo, ognuno dei precedenti tre eventi ha probabilità $\frac{1}{3}$ di verificarsi. Abbiamo quindi $\forall h = 1, \dots, k-1$:

$$\begin{cases} P[X_i = h+1 | X_{i-1} = h] = \frac{1}{3} \\ P[X_i = h-1 | X_{i-1} = h] = \frac{1}{3} \\ P[X_i = h | X_{i-1} = h] = \frac{1}{3} \end{cases}$$

Che ci fornisce la seguente catena (con i dovuti accorgimenti per $h = 0$)



Quello che vogliamo calcolare è $f_h = E[T|X_0 = h]$ al fine di capire come settare il numero di iterazioni t e magari ottenere anche correttezza *w.h.p.*

$$\begin{aligned} f_h = E[T|X_0 = h] =& E[T|X_0 = h, X_1 = h+1]P[X_1 = h+1|X_0 = h] \\ &+ E[T|X_0 = h, X_1 = h-1]P[X_1 = h-1|X_0 = h] \\ &+ E[T|X_0 = h, X_1 = h]P[X_1 = h|X_0 = h] = \\ &= 1 + \frac{f_{h+1} + f_{h-1} + f_h}{3} \end{aligned}$$

Otteniamo il sistema di $k+1$ equazioni in $k+1$ incognite:

$$\begin{cases} f_h = 1 + \frac{f_{h+1}+f_{h-1}+f_h}{3}, \ \forall h = 1, \dots, k-1 \\ f_k = 0 \\ f_0 = 1 + \frac{f_0+2f_1}{3} \end{cases}$$

Osserviamo che $f_h \geq f_{h+1}$ e ricaviamo dalla prima equazione del sistema:

$$f_h - f_{h+1} = 3 + f_{h-1} - f_h$$

Definiamo $\Delta_h := f_h - f_{h+1}$, allora:

$$\Delta_h = 3 + \Delta_{h-1} = 3 + (3 + \Delta_{h-2}) = \cdots = 3h + \Delta_0$$

Dalla terza equazione del sistema si ricava $\Delta_0 = f_0 - f_1 = \frac{3}{2}$, quindi:

$$\Delta_h = 3h + \frac{3}{2}$$

Riscriviamo adesso f_h come:

$$f_h = (f_h - f_{h+1}) + (f_{h+1} - f_{h+2}) + \cdots + (f_{k-1} - f_k) + f_k =$$
$$= f_k + \sum_{i=h}^{k-1} \Delta_i = \sum_{i=0}^{k-1} \Delta_i - \sum_{i=0}^{h-1} \Delta_i = \frac{3k^2 - 3h^2}{2}$$

Nel caso peggiore, quando $h = 0$, sono necessari $\frac{3k^2}{2}$ cambiamenti in valore atteso. Dalla *disuguaglianza di Markov* con $a = 3n^2$:

$$P[T \geq 3n^2] \leq \frac{\frac{3k^2}{2}}{3n^2} \leq \frac{\frac{3n^2}{2}}{3n^2} = \frac{1}{2}$$

Scegliamo dunque $t = 3n^2 b$ con $b \in \mathbb{N}$ per avere $P[Alg\ sbaglia] \leq 2^{-b}$, quindi se scegliamo $b = \lceil c \log n \rceil$, $c > 0$, otteniamo la correttezza *w.h.p.*

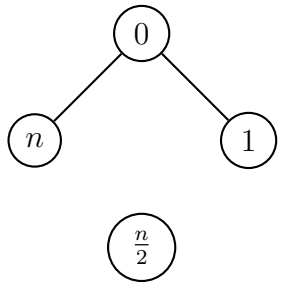
Nel ciclo *for* è fatta un'enumerazione e controllo dei possibili triangoli di G che richiede tempo $\mathcal{O}(n^3)$, avremo un *Running Time*:

$$Time(n) = \mathcal{O}(n^5 \log n)$$

Osservazione: nell'analisi dell'algoritmo abbiamo supposto che il grafo di input è 3-colorabile e abbiamo fissato una sua 3-colorazione g, tuttavia nulla vieta che tale grafo possa avere più di una 3-colorazione. Per alcuni grafi potrebbe accadere che troviamo il 2-labeling prima di raggiungere lo stato goal della 3-colorazione g fissata, dato che il 2-labeling f che costruiamo potrebbe essere in accordo con qualche altra 3-colorazione diversa da g.

Esercizio 3

Alla prima lettura del problema ho pensato che i nodi su cui la pedina si ferma con maggiore probabilità sono i nodi che si trovano intorno al nodo etichettato con $\frac{n}{2}$, cioè i nodi che si troverebbero a "metà percorso" dell'anello. Sarebbe abbastanza controintuitivo immaginare che la pedina si fermi spesso sui nodi immediatamente adiacenti al nodo 0, infatti, uno dei due nodi adiacenti a 0 è sicuramente visitato all'istante $t = 1$ e all'istante $t = 2$ abbiamo probabilità pari a $\frac{1}{2}$ di tornare in 0 al fine di ottenere un'altra chance per spostare la pedina nell'altro nodo adiacente non ancora visitato.



Data la *v.a.* X, definita come riportato nel testo:

$$X := \textit{ nodo su cui si ferma la pedina}$$

ho provato ad ipotizzare la sua distribuzione di probabilità ma non ho avuto molto successo... Sono abbastanza sicuro che X non segua una *distribuzione binomiale* poichè risulta un po' difficile descriverla in termini di k successi su n tentativi: è fuorviante fissare un numero di tentativi da eseguire in un processo aleatorio di questo tipo, similmente, parlare di k successi non mi sembra abbia molto senso. La *distribuzione geometrica* sembra intuitivamente più adatta ma in questo caso la sfida sarebbe definire la probabilità di successo/insuccesso.

Fatte queste osservazioni iniziali, non ho per ora una vera intuizione su quale possa essere la distribuzione di probabilità di X.

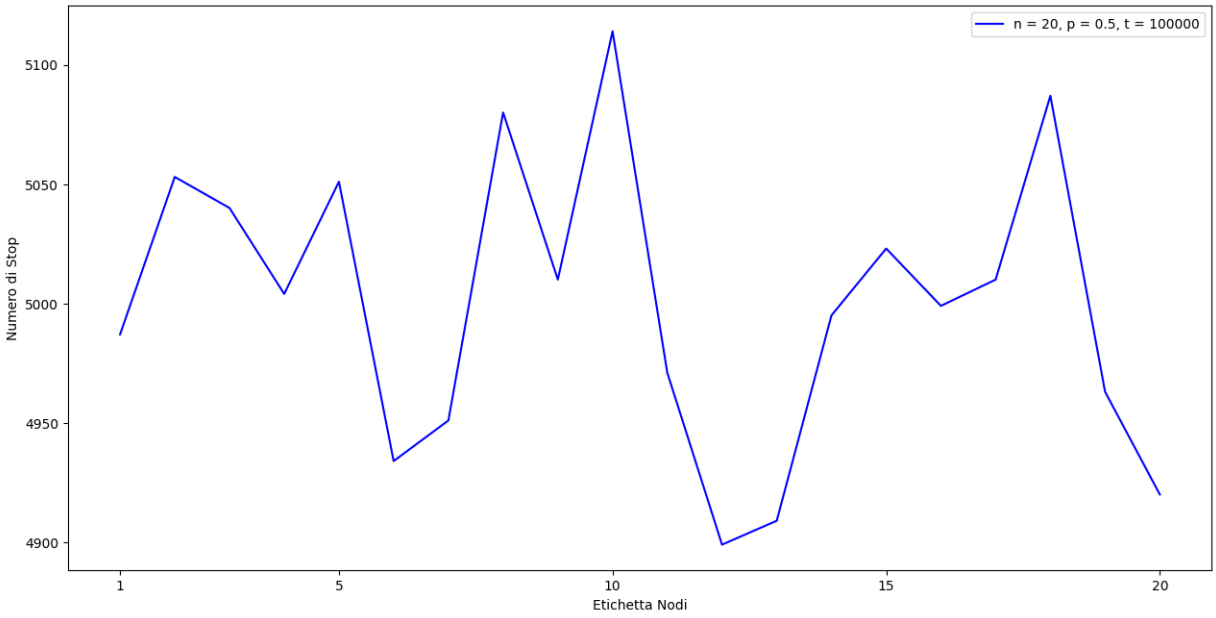
3.1 e 3.2

Gli esercizi implementativi sono stati realizzati in Python e si trovano nella cartella *Esercizio 3* suddivisi nei vari punti. Le funzioni principali sono commentate per cercare di rendere chiaro ogni dettaglio rilevante.

3.3

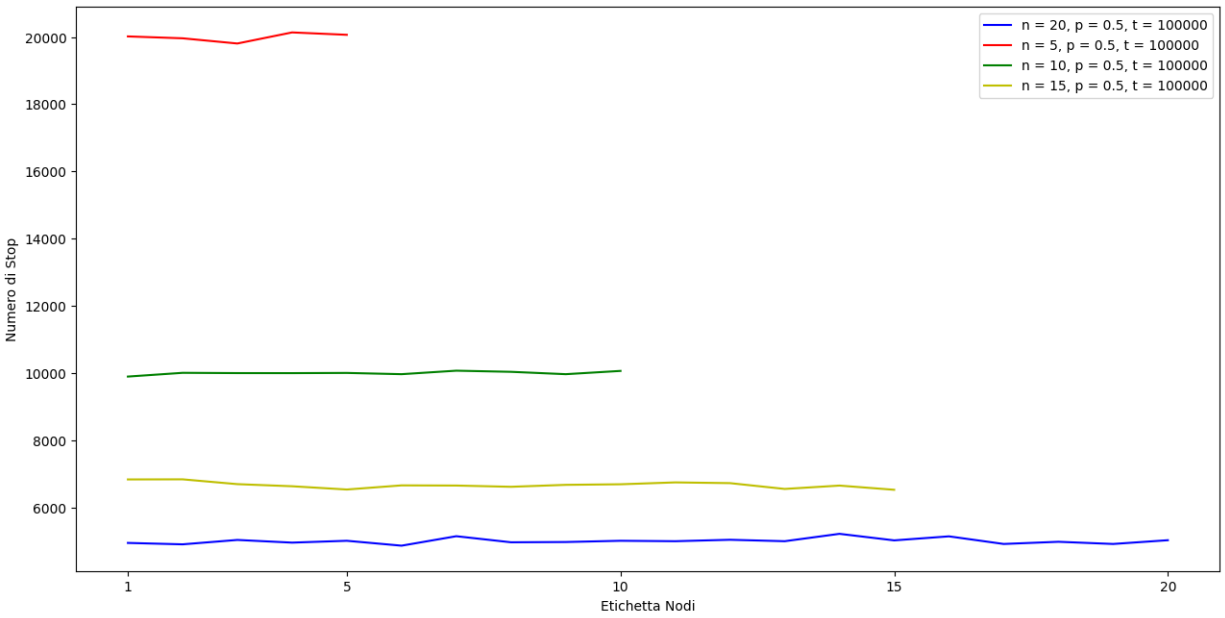
In questa sezione si esplorano i dati ottenuti dalla simulazione del processo aleatorio su diversi valori del numero di nodi n. I dati verranno riportati sotto forma di grafici ottenuti dalla funzione *makePlot* definita all'interno del file *3_2.py*, la funzione si trova in un commento per evitare che venga eseguita ogni volta all'esecuzione del file.

La probabilità di spostamento tra i nodi è fissata a $p = \frac{1}{2}$ e il numero di iterazioni della simulazione di un processo è fissato a $t = 10^5$. L'unico parametro che varia è il numero di nodi n. Sull'asse delle ascisse troviamo le etichette dei nodi, sull'asse delle ordinate troviamo il numero di volte che la pedina si è fermata su un certo nodo. Osserviamo il seguente grafico ottenuto per $n = 20$:



Dal grafico si nota che il numero di volte che la pedina si ferma su un nodo rimane relativamente concentrato attorno 5000, per questo grafico tra 4900 e 5100 (circa) per ogni nodo dell'anello.

Con il prossimo grafico si guardano le cose un po' più dall'alto, mettendo a confronto l'andamento di queste rette per diversi valori di n ed è proprio da questo grafico che ho avuto la vera intuizione su quale potrebbe essere la distribuzione di X.



Dai dati emerge che la pedina si ferma su ogni nodo più o meno lo stesso numero di volte indipendentemente da n. Questo suggerisce che la $v.a.$ X segue una *distribuzione uniforme*. L'intuizione di partenza, secondo la quale i nodi vicini al nodo 0 sono i nodi su cui la pedina si ferma meno frequentemente, è quindi errata.

3.4

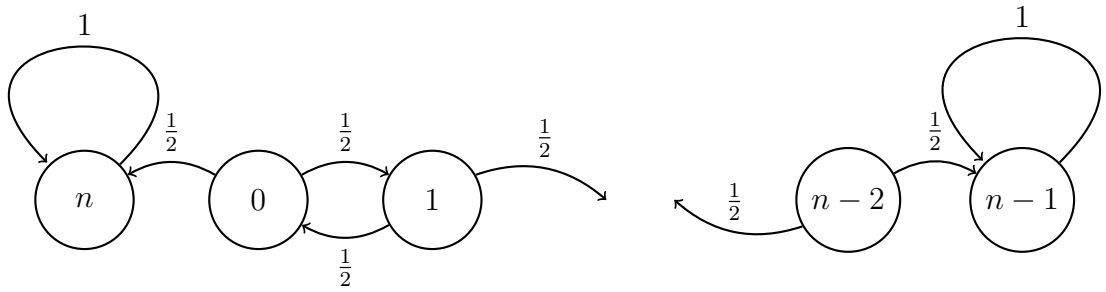
Proviamo a capire quale è la probabilità che la pedina si ferma sul nodo n. La supposizione da cui sono partito è che la probabilità che la pedina si ferma sul nodo n coincide con la probabilità che la pedina attraversa tutti gli altri nodi dell'anello senza mai passare prima per n. In questo caso particolare che stiamo considerando, la pedina deve arrivare ad $n-1$ senza toccare il nodo n. Abbiamo quindi due situazioni diverse:

1) la pedina, partendo da 0, arriva al nodo etichettato con $n-1$ senza passare prima per n;

2) la pedina, partendo da 0, passa sul nodo etichettato con n prima di arrivare sul nodo etichettato con $n-1$.

Torna utile definire la v.a. booleana $Y :=$ *la pedina arriva in n-1 prima di passare per la prima volta su n.* Definiamo anche $p_h = P[Y = 1|X_0 = h]$ $\forall h = 0, 1, \ldots, n-1, n$.

La probabilità che vogliamo calcolare è relativa al primo caso e la indichiamo con $p_0 = P[Y = 1|X_0 = 0]$, inoltre, dalla definizione di Y segue che $p_{n-1} = 1$ e $p_n = 0$. Riconduciamo il problema all'analisi della seguente catena di Markov con probabilità di transizione pari a $\frac{1}{2}$ per ogni nodo diverso da n e $n-1$



Da cui il sistema:

$$\begin{cases} p_h = \frac{p_{h+1}+p_{h-1}}{2}, \ \forall h = 0, \dots, n-2 \\ p_n = 0 \\ p_{n-1} = 1 \end{cases}$$

Con l'osservazione $p_{h+1} \geq p_h$ e il calcolo dei $\Delta_{h+1} = p_{h+1} - p_h$, si ottiene:

$$p_{h+1} = p_n + (h+2)p_0$$

Sostituendo $h = n-2$ e riscrivendo l'equazione si ottiene:

$$p_0 = \frac{p_{n-1}}{n} = \frac{1}{n}$$

Con lo stesso procedimento e qualche modifica alle etichette della catena, si può calcolare la probabilità che la pedina si ferma sul nodo 1 che è sempre $\frac{1}{n}$. Per i restanti nodi che non sono adiacenti al nodo di partenza ciò che si vuole calcolare è la stessa cosa, ovvero si vuole sapere la probabilità che la pedina visita tutti i nodi dell'anello prima di raggiungere il nodo i. In questi altri casi il calcolo si complica poichè ci sono più situazioni da considerare, infatti la pedina potrebbe raggiungere prima il nodo $i-1$ e poi "tornare indietro" fino al nodo $i+1$ o viceversa. Tuttavia non sono riuscito a generalizzare la catena e il calcolo dei p_0.