Nom : Prénom :

CONCOURS BLANC INFORMATIQUE – TSI2 – 2022

Pour répondre à une question, il est possible de faire appel aux fonctions définies dans les questions précédentes.
Si vous êtes amené à repérer ce qui peut vous sembler être une erreur d'énoncé, vous le signalerez sur votre copie et devrez poursuivre votre composition en expliquant les raisons des initiatives que vous avez été amené à prendre.

L'algorithme PageRank de Google

Cela fait plus de 20 ans que Google domine le marché des moteurs de recherche. Cette longévité semble indiquer une réelle pertinence des résultats fournis.
Depuis sa conception en 1998, l'algorithme de recherche de Google continue d'évoluer et la plupart des améliorations demeurent des secrets industriels. L'idée principale, par contre, est connue puisqu'elle est l'objet d'une célèbre publication de Sergueï Brin et Larry Page (les cofondateurs) : *The anatomy of a large-scale hypertextual web search engine. Stanford University, 1998*. Le fonctionnement de PageRank, l'algorithme au centre du moteur de recherche, y est détaillé.

I. Une base de données du Web ?

Imaginons qu'il existe un registre complet de l'internet structuré sous la forme d'une base de données relationnelle.
On supposera dans la suite que chaque nouvelle page web créée entraîne un enregistrement sur une base de données constituée des 3 tables décrites ci-dessous :

table 1	
site	TEXT
page	TEXT
creation	TEXT

table 2	
index	INT
page	TEXT
liensortant	TEXT

table 3	
index	INT
page	TEXT
keyword	TEXT

Le contenu des trois tables pour un internet lilliputien imaginaire :

a.io	a.io/index.html	2022-08-25
a.io	a.io/1/index.html	2022-08-30
a.io	a.io/2/index.html	2022-08-30
b.com	b.com/index.html	2002-11-25
b.com	b.com/1/index.html	2004-05-02
b.com	b.com/2/index.html	2004-05-03
c.fr	c.fr/index.html	2020-10-04
c.fr	c.fr/1/index.html	2020-10-04
d.edu	d.edu/index.html	1998-03-03

1	a.io/index.html	a.io/1/index.html
2	a.io/index.html	a.io/2/index.html
3	a.io/1/index.html	b.com/index.html
5	b.com/index.html	b.com/1/index.html
7	b.com/index.html	b.com/2/index.html
8	b.com/1/index.html	b.com/2/index.html
9	b.com/1/index.html	c.fr/index.html
4	a.io/1/index.html	d.edu/index.html
10	b.com/2/index.html	d.edu/index.html
11	c.fr/index.html	c.fr/1/index.html
12	c.fr/index.html	b.com/2/index.html
13	d.edu/index.html	a.io/1/index.html
6	b.com/index.html	d.edu/index.html
14	b.com/1/index.html	a.io/index.html

1	a.io/index.html	base
2	a.io/index.html	SQL
3	a.io/1/index.html	Gauss
4	a.io/1/index.html	Lagrange
5	a.io/1/index.html	Python
6	b.com/index.html	Euler
7	d.edu/index.html	base
8	c.fr/1/index.html	base
10	a.io/2/index.html	base
11	a.io/1/index.html	base

1. Que désignent les deux colonnes dans les trois premiers tableaux (présentant les tables) ?

colonne 1 :

colonne 2 :

2. Quel(s) attribut(s) et/ou association(s) d'attributs de la table 1 peut/peuvent-il(s) servir de **clé primaire** ?

clé(s) primaire(s) possible(s) :

3. Écrivez une requête SQL qui permet d'obtenir les **pages** créées avant 2010 en se limitant à 3 résultats.

4. Écrivez une requête SQL permettant de compter, pour chaque **site**, le **nombre de pages** qu'il contient (les en-têtes de la table affichée devront être « `site` » et « `nb pages` »).

5. Écrivez une requête SQL n'affichant que les **pages** ayant au moins 2 liens sortants ainsi que leur **nombre de liens sortants**.

6. Écrivez une requête SQL qui permet d'obtenir les **sites** et les **pages** contenant un lien vers `d.edu/index.html`.

7. Écrivez une requête SQL calculant, pour chaque **page**, sa « **popularité** », où la popularité est définie comme le nombre de liens pointant vers la page (on supposera pour simplifier que chaque page a au moins un lien pointant vers elle).

8. Écrivez une requête SQL qui classe, pour le mot de clé « base », les **pages** où il est présent, dans l'ordre décroissant de leur popularité telle que définie à la question précédente.

Comme un tel registre centralisé n'existe pas, c'est à la charge du moteur de recherche d'explorer le web pour produire un annuaire du même type que notre base de données imaginaire.

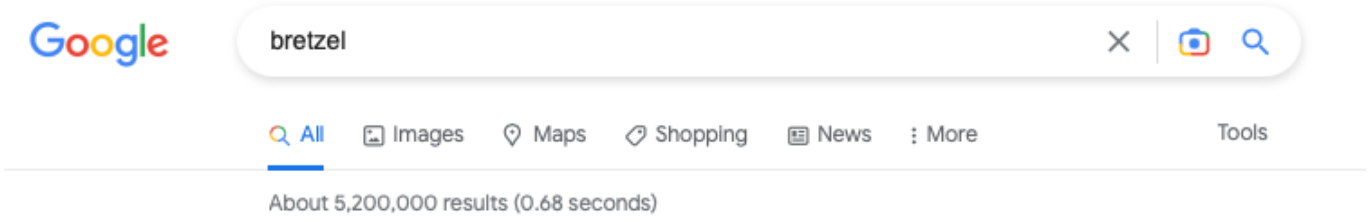
9. Wikipedia en langue anglaise compte environ 6,5 millions d'article en octobre 2022. Estimez les tailles des trois tables si la base de données précédente était construite uniquement à partir de ces articles. Vous veillerez à expliciter vos hypothèses.

taille table1 : justification :

taille table2 :	justification :

taille table3 :	justification :

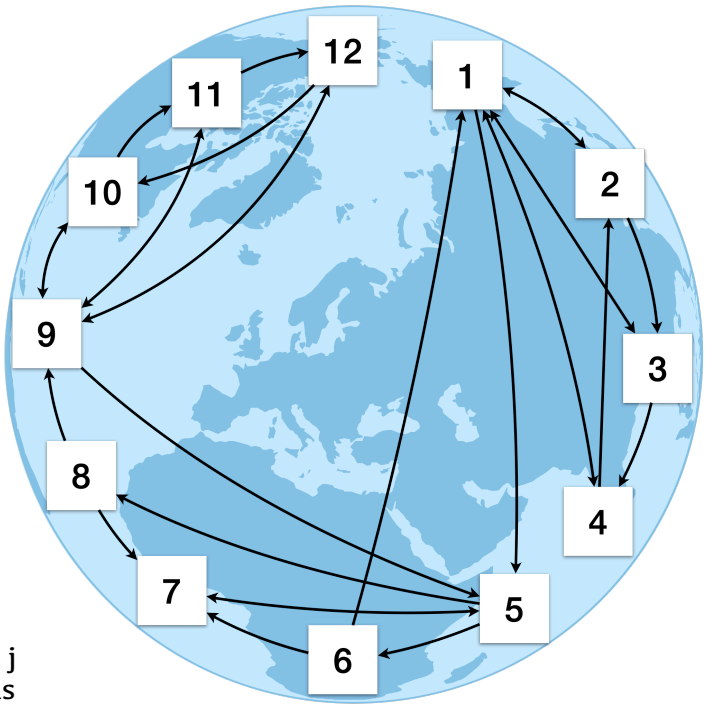
Pour un mot clé donné, il y a typiquement des millions de pages correspondantes que le moteur de recherche va devoir classer.



II. Graphe du Web

Profitons du peu de structure disponible : le web n'est pas une collection de textes indépendants mais un immense hypertexte où les pages se citent mutuellement.
En négligeant le contenu des pages pour se concentrer sur les liens entre elles, on obtient la structure d'un graphe.

Exemple :



Attention,
on s'autorise ici
des doubles flèches ↔
entre deux sommets i et j
pour représenter à la fois
un un arc $i \to j$
et un arc $j \to i$.

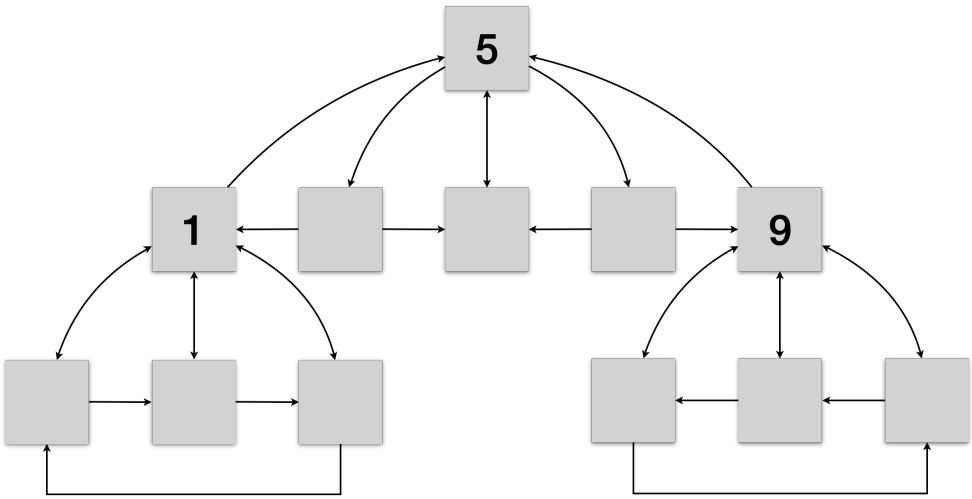
Dans la suite, on note $P_1, P_2, \ldots, P_n$ les pages web et $j \to i$ si la page P_j cite la page P_i.

10. Le graphe obtenu est-il (rayer la mention inutile) :

orienté	/	non orienté
acyclique	/	cyclique
pondéré	/	non pondéré

Comment exploiter ce graphe ?
L'idée de départ est de considérer qu'un lien $j \to i$ correspond à une recommandation de la part de la page P_j d'aller lire la page P_i. Dans cette hypothèse, le lien est une sorte de vote de confiance de la page P_j en faveur de l'autorité de la page P_i.

Présentons le graphe précédent de manière à faire apparaître une **hiérarchie** possible :



11. Complétez les cases laissées vierges.

Parmi les pages P_1, P_2, P_3, P_4, la page P_1 sert de référence commune et semble un bon point de départ pour chercher des informations. Même chose avec les groupes $P_9, P_{10}, P_{11}, P_{12}$, où P_9 sert de référence commune, et P_5, P_6, P_7, P_8, où P_7 est la plus citée.
Et comme P_1 et P_9, déjà reconnues comme importantes, font référence à la page P_5, on pourrait soupçonner que la page P_5 contient de l'information essentielle pour l'ensemble, qu'elle est la plus pertinente.
Selon ce raisonnement, **un modèle de classification des pages se doit de faire apparaître cette hiérarchie**. Nous allons en faire par la suite notre critère de réussite…

A. Premier modèle : comptage naïf

Il est plausible qu'une page importante reçoive beaucoup de liens. Avec un peu de naïveté, on croira aussi la réciproque : si une page reçoit beaucoup de liens, alors elle est importante.
On pourrait ainsi définir l'**importance** μ_i de la page P_i comme le nombre de liens $j \to i$:

$$\mu_i = \sum_{j \to i} 1$$

Autrement dit, μ_i est le nombre de « votes » pour la page P_i, où chaque vote contribue pour la même valeur 1.

12. Écrivez en python une fonction `imp1` renvoyant l' « importance » μ_i d'une page.
    On lui passe en paramètre le graphe du web tel que présenté ci-dessus sous la forme d'une liste d'adjacence implémentée par un dictionnaire.
    Dans notre exemple : `web = {1:[2,3,4,5],2:[1,3], 3:[1,4],4:[1,2],5:[6,7,8],6:[1,7],7:[5],8:[7,9],9:[5,10,11,12],10:[9,11],11:[9,12],12:[9,10]}`

```
def imp1(i: int, web: dict) -> int:
```

13. En théorie des graphes, comment appelle-t-on μ_i pour le sommet i ?

14. On a déjà calculé l'importance d'une page selon le même modèle dans la partie 1.
    Comment l'avions-nous alors appelée ?

15. Que valent les importances suivantes dans notre exemple :

$\mu_1 =$ $\mu_5 =$ $\mu_7 =$ $\mu_9 =$

Bien que cette définition de l'importance soit claire et facile à calculer, on constate sur notre exemple qu'elle ne reproduit pas la hiérarchie attendue entre les pages.
Pire, ce comptage naïf est trop facile à manipuler par quiconque souhaitant gonfler artificiellement l'importance d'une de ses pages.

16. Comment ferait-il ?

B. Deuxième modèle : comptage pondéré

On peut supposer que les pages émettant beaucoup de liens sont des prescripteurs moins sélectifs.
On va alors diminuer la confiance apportée à leurs recommandations.
Pour diminuer leur poids, on va partager le vote de la page P_j en ℓ_j parts égales, où ℓ_j désigne le nombre de liens émis par la page P_j. On définit ainsi une mesure plus fine de l'importance :

$$\mu_i = \sum_{j \to i} \frac{1}{\ell_j}$$

17. Écrivez en python une fonction `imp2` retournant l'importance μ_i ainsi définie.

```
def imp2(i: int, web: dict) -> float:
```

18. Que valent maintenant les importances suivantes dans notre exemple :

$\mu_1 =$ $\mu_5 =$ $\mu_7 =$ $\mu_9 =$

On voit donc que la formule peine encore à reproduire la hiérarchie attendue entre les pages.
Et comme avant, ce comportement est trop facile à truquer.

Un **graphe dense** est un graphe à n sommets ayant $O(n)$ arêtes par sommet, alors qu'un graphe creux n'a que $O(1)$ arêtes par sommet.

19. Selon vous, le graphe du web est-il dense ou creux ?

20. Quelle est alors la complexité des fonctions `imp1` et `imp2` en fonction de n ?

21. Pourquoi représenter le graphe par une liste d'adjacence plutôt que par une matrice d'adjacence.

C. Troisième modèle : comptage récursif

Heuristiquement, une page P_i paraît importante si beaucoup de pages importantes la citent. Ceci nous mène à définir l'importance μ_i de manière récursive comme suit :

$$\mu_i = \sum_{j \to i} \frac{1}{\ell_j} \mu_j$$

Ici, le poids du vote $j \to i$ est proportionnel au poids μ_j de la page émettrice.

C'est facile à formuler, mais moins évident à calculer…

Comme il s'agit d'un système de n équations à n inconnus, on peut penser à utiliser la méthode du **pivot de Gauss** pour le résoudre.

Pour notre exemple, la matrice augmentée représentant le système est donné ci-dessous :

$$\left(\begin{array}{cccccccccccccc}
-1 & 1/2 & 1/2 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/4 & -1 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/4 & 1/2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/4 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/4 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 1/4 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \ldots & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/3 & 1/2 & -1 & 1/2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & -1 & 1/2 & 1/2 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & -1 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 0 & 1/2 & -1 & 0
\end{array}\right)$$

22. Complétez la valeur manquante dans la matrice.

On peut vérifier que le vecteur $\mu = (2,1,1,1,3,1,2,1,2,1,1,1)^\top$ est solution du système.
Enfin ! La page P_5 est repérée comme la plus importante, suivie de P_1, P_7 et P_9. C'est encourageant !

Le code de la fonction permettant de transformer la matrice précédente en matrice échelonnée est donné ci-contre.

23. Reste à coder la fonction de recherche du pivot.
On utilise ici, la **recherche du pivot partiel** : votre fonction doit, à partir d'une ligne h et d'une colonne k (h et k données en paramètre, trouver la ligne i de l'élément $m_{i,k}$ de la matrice M (elle aussi donnée en paramètre) de plus grande valeur absolue, pour i allant de h à n.

```
def Gauss(M,recherchePivot) :
    n = len(M)
    h = k = 0
    tol = 1e-9
    while h < n and k < n+1 :
        ipivot = recherchePivot(M,h,k)
        pivot = M[ipivot][k]
        if abs(pivot) < tol :
            k += 1
        else :
            if h != ipivot :
                M[h],M[ipivot] = M[ipivot],M[h]
            for j in range(k,n+1) :
                M[h][j] /= pivot
            for i in range(h+1,n) :
                f = M[i][k]
                for j in range(k,n+1) :
                    M[i][j] -= M[h][j] * f
            h += 1
            k += 1
```

```
def rechercheePivotPartiel(M: list, h: int, k: int) -> int:
```

On obtient la matrice échelonnée suivante :

```
[[1.0, -0.5, -0.5, -0.5, -0.0, -0.5, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0],
 [0.0, 1.0, -0.14285714285714285, -0.7142857142857143, -0.0, -0.14285714285714285, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0],
 [0.0, 0.0, 1.0, -0.7272727272727273, -0.0, -0.2727272727272727, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0],
 [0.0, 0.0, 0.0, 1.0, -3.142857142857143, 0.5714285714285714, 3.142857142857143, 0.0, 0.7857142857142857, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 1.0, -0.5000000000000001, -1.0, -0.0, -0.25, -0.0, -0.0, -0.0, -0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, -0.4, -0.0, -0.1, -0.0, -0.0, -0.0, -0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, -1.25, -0.37500000000000006, -0.0, -0.0, -0.0, -0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, -0.5, -0.0, -0.0, -0.0, -0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, -0.6666666666666666, -0.6666666666666666, -0.6666666666666666, -0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, -0.19999999999999998, -0.7999999999999999, -0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, -0.9999999999999997, -0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -3.3306690738754696e-16, 0.0]]
```

24. Parmi les 12 équations du système, combien sont-elles indépendantes ?

25. Quelle est la complexité de la fonction `Gauss` ?

L'algorithme PageRank utilise une méthode plus simple à implémenter et bien plus rapide !

Pour développer une intuition de la méthode, imaginons un surfeur aléatoire qui se ballade sur internet en cliquant sur les liens au hasard. Comment évolue sa position ?
À titre d'exemple, supposons que notre surfeur démarre au temps $t = 0$ sur la page P_7. Le seul lien pointe vers P_5, donc au temps $t = 1$, le surfeur s'y retrouve avec une probabilité 1. De P_5 partent trois liens, donc au temps $t = 2$, il se trouve sur une des 3 pages P_6, P_7, P_8, avec une probabilité d'$1/3$. En continuant ainsi, on obtient :

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
t = 0	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000
t = 1	0,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
t = 2	0,000	0,000	0,000	0,000	0,000	0,333	0,333	0,333	0,000	0,000	0,000	0,000
t = 3	0,167	0,000	0,000	0,000	0,333	0,000	0,333	0,000	0,167	0,000	0,000	0,000
t = 4	0,000	0,042	0,042	0,042	0,417	0,111	0,111	0,111	0,000	0,042	0,042	0,042
t = 5	0,118	0,021	0,021	0,021	0,111	0,139	0,250	0,139	0,118	0,021	0,021	0,021
	...	...	...	...	...	...	...	...	...	...	...	...
t = 29	0,117	0,059	0,059	0,059	0,177	0,059	0,117	0,059	0,117	0,059	0,059	0,059
t = 30	0,117	0,059	0,059	0,059	0,177	0,059	0,117	0,059	0,117	0,059	0,059	0,059

On observe une diffusion qui **converge** assez rapidement vers une **distribution stationnaire**.

Vérifions-le en partant d'une autre page, disons P_1 :

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
t = 0	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
t = 1	0,000	0,250	0,250	0,250	0,250	0,000	0,000	0,000	0,000	0,000	0,000	0,000
t = 2	...	...	...	...	...	...	...	...	...	...	...	...
t = 3	0,229	0,156	0,156	0,156	0,177	0,000	0,083	0,000	0,042	0,000	0,000	0,000
t = 4	0,234	0,135	0,135	0,135	0,151	0,059	0,059	0,059	0,000	0,010	0,010	0,010
t = 5	0,233	0,126	0,126	0,126	0,118	0,050	0,109	0,050	0,045	0,005	0,005	0,005
	...	...	...	...	...	...	...	...	...	...	...	...
t = 69	0,117	0,059	0,059	0,059	0,177	0,059	0,117	0,059	0,117	0,059	0,059	0,059
t = 70	0,117	0,059	0,059	0,059	0,177	0,059	0,117	0,059	0,117	0,059	0,059	0,059

26. Complétez la ligne correspondant à $t = 2$.

Bien que la diffusion mette plus de temps, la mesure stationnaire est la même ! Elle coïncide d'ailleurs avec notre solution $\mu = (2,1,1,1,3,1,2,1,2,1,1,1)^\top$, ici divisée par 17 pour normaliser la somme à 1.
Les pages où μ_i est grande sont par le fait les plus « fréquentées » ou les plus « populaires ». Dans la quête de classer les pages web, c'est encore un argument pour utiliser la mesure μ comme indicateur.

Comment formaliser cette diffusion ? Supposons qu'au temps t, notre surfeur aléatoire se trouve sur la page P_j avec une probabilité p_j. La probabilité de partir de P_j en suivant le lien $j \to i$ est alors $\frac{1}{\ell_j} p_j$.

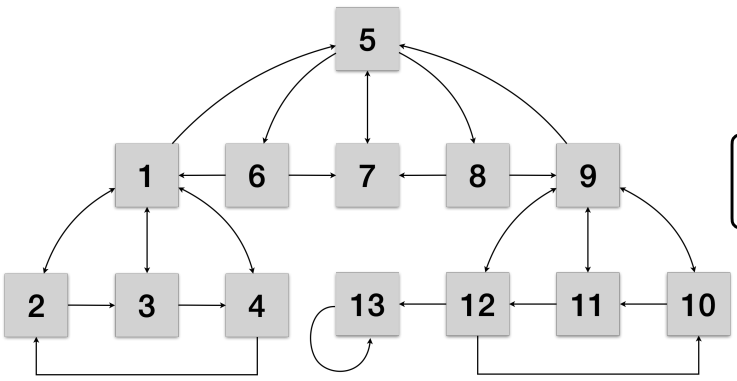
La probabilité d'arriver au temps $t+1$ sur la page P_i est donc :

$$p'_i = \sum_{j \to i} \frac{1}{\ell_j} p_j$$

Étant donné la distribution initiale p, la **loi de transition** ci-dessus définit la distribution $p' = T(p)$. C'est ainsi que l'on obtient la ligne $t+1$ à partir de la ligne t dans nos exemples (en théorie des probabilité, cela s'appelle une **chaîne de Markov**). La **mesure stationnaire** est caractérisée par l'équation d'équilibre $\mu = T(\mu)$ qui est justement notre équation $\mu_i = \sum_{j \to i} \frac{1}{\ell_i} \mu_j$.

Mais il existe un obstacle à ce modèle : les **trous noirs**...

27. Trouvez la solution stationnaire (le vecteur μ) dans l'exemple ci-dessous.



$\mu =$

Pour échapper aux trous noirs, Google utilise un modèle plus raffiné : avec une probabilité fixée c, le surfeur abandonne sa page actuelle P_j et recommence sur une des n pages du web, choisie de manière équiprobable ; sinon, avec une probabilité $1-c$, le surfeur suit un des liens de la page P_j, choisi lui aussi de manière équiprobable.
Cette astuce de « **téléportation** » évite de se faire piéger par une page sans issue, et garantit d'arriver n'importe où dans le graphe, indépendamment des questions de connexité.

Dans ce modèle, la transition est donnée par :

$$p'_i = \frac{c}{n} + \sum_{j \to i} \frac{1-c}{\ell_j} p_j$$

Le premier terme $\frac{c}{n}$ vient de la téléportation et le second terme de la marche aléatoire précédente.
La mesure d'équilibre vérifie donc :

$$\mu_i = \frac{c}{n} + \sum_{j \to i} \frac{1-c}{\ell_j} \mu_j$$

Le paramètre c reste à calibrer. Pour $c = 0$, on retrouve le modèle précédent. Pour $0 \leq c \leq 1$, la valeur $1/c$ est le nombre moyen de pages visitées, c'est-à-dire le nombre de liens suivis plus un, avant de recommencer sur une page aléatoire (processus de Bernoulli). Par exemple, le choix $c = 0{,}15$ correspond à suivre environ 6 liens en moyenne, ce qui semble une description réaliste.

Reprenons l'exemple initial et simulons la marche aléatoire partant de la page P_1 :

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
t = 0	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
t = 1	0,013	0,225	0,225	0,225	0,225	0,013	0,013	0,013	0,013	0,013	0,013	0,013
t = 2	0,305	0,111	0,111	0,111	0,028	0,076	0,087	0,076	0,034	0,020	0,020	0,020
t = 3	0,186	0,124	0,124	0,124	0,158	0,021	0,085	0,021	0,071	0,028	0,028	0,028
t = 4	0,180	0,105	0,105	0,105	0,140	0,057	0,075	0,057	0,057	0,040	0,040	0,040
t = 5	0,171	0,095	0,095	0,095	0,126	0,052	0,101	0,052	0,087	0,042	0,042	0,042
	...	...	...	...	...	...	...	...	...	...	...	...
t = 29	0,120	0,066	0,066	0,066	0,150	0,055	0,102	0,055	0,120	0,066	0,066	0,066
t = 30	0,120	0,066	0,066	0,066	0,150	0,055	0,102	0,055	0,120	0,066	0,066	0,066

La mesure stationnaire est vite atteinte, et la page P_5 arrive en tête avec $\mu_5 = 0{,}15$ avant les pages P_1 et P_9 avec $\mu_1 = \mu_9 = 0{,}12$.

28. Codez un algorithme itératif permettant d'obtenir la mesure stationnaire μ. Votre fonction prendra en paramètre le graphe du web sous la forme d'une liste d'adjacence implémentée par un dictionnaire et retournera la mesure stationnaire μ sous la forme d'une liste de n flottants.
    Pour $t = 0$, vous initialiserez la liste correspondant au vecteur μ avec une valeur de c/n pour chaque élément.
    Votre implémentation se limitera à un graphe du web calqué sur le modèle de notre exemple où chaque page est désigné par un entier entre 1 et n.

```
def PageRank(web: dict) -> list:
```

La loi de transition définit une application $T : p \mapsto p'$ qui est contractante de rapport $1 - c$. Le théorème du point fixe de Banach nous assure alors que l'équation $\mu_i = \frac{c}{n} + \sum_{j \to i} \frac{1-c}{\ell_j} \mu_j$ admet une unique solution μ vérifiant $\sum_{i=1}^{n} \mu_i = 1$ où $\forall i, \mu_i \geq 0$. Pour toute distribution de probabilité initiale, le processus de diffusion converge vers cette solution stationnaire μ. La convergence est au moins aussi rapide que celle de la suite géométrique $(1 - c)^n$ vers 0.

29. De quoi la démonstration de l'existence de la solution stationnaire μ permet-elle de s'assurer par rapport à l'algorithme PageRank ?

30. Quelle méthode algorithmique permettant de calculer l'état d'un système au temps $t + h$ en fonction de son état au temps t présente-t-elle un potentiel problème d'**instabilité numérique** ?

Ici, pas de soucis ! Le fait que T soit contractante nous assure la stabilité numérique. Les erreurs d'arrondi liées à l'usage de nombres à virgule flottante ne sont pas amplifiées dans les calculs itératifs.

Ce sujet est une adaptation de l'article « L'algorithme PageRank de Google : une promenade sur la toile » de Michael Eisermann.