Google PageRank

```
Needs["GraphUtilities`"]
```

Introduction

$$p^T = p^T (\alpha S + (1 - \alpha) E)$$

Ci-dessus l'équation au coeur de l'algorithme PageRank. A cette étape nous présentons cette équation pour des raisons seulement esthétiques.
Il s'agira, au cours de l'exposé à suivre, d'en approcher le sens.
Cette équation modélise l'idée suivante : **une page Web est importante si des pages Web importantes pointent vers elle**.
Nous allons voir que ces scores d'importance sont en fait les valeurs stationnaires d'une très grosse chaîne de Markov.
Brin et Page ont construit cet algorithme à la fin des années 90 (un brevet a été déposé en 1998 et validé en 2001) sans que le lien précis avec la théorie des chaînes de Markov n'ait été établi. Ce lien est important car il permet en particulier d'obtenir des garanties théoriques (e.g., vitesse de convergence, etc.), mais il n'est pas absolument nécessaire à la compréhension de la dérivation de l'algorithme.

On note $r(P_i)$ le score de la page P_i, B_{P_i} l'ensemble des pages qui pointent ("backlink") vers P_i.

Exprimer $r(P_i)$ en fonction du score des pages qui pointent vers P_i.

$$r(P_i) = \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|P_j|}$$

Où $|P_j|$ représente le nombre de liens sortants pour la page j.
Nous rencontrons un problème : les valeurs $r(P_j)$ nécessaires au calcul de $r(P_i)$ nous sont inconnues.

Proposer un algorithme pour le calcul de $r[P_i]$.

A partir de l'équation précédente, nous proposons un processus itératif naïf afin de calculer les scores des pages. Bien entendu, pour l'instant rien n'assure la convergence de ce processus. Et quand même convergerait-il, serions-nous certains que les scores attribués aux pages par la configuration stable modélisent correctement la notion subjective de l'importance d'une page ?

$$r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|P_j|}$$

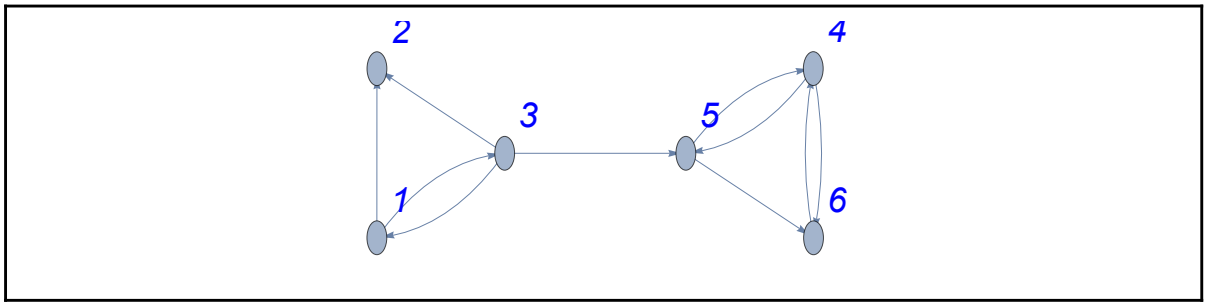
$$r_0(P_i) = \frac{1}{n}$$

Avec n le nombre de pages Web indexées.

Calculer quelques itérations.

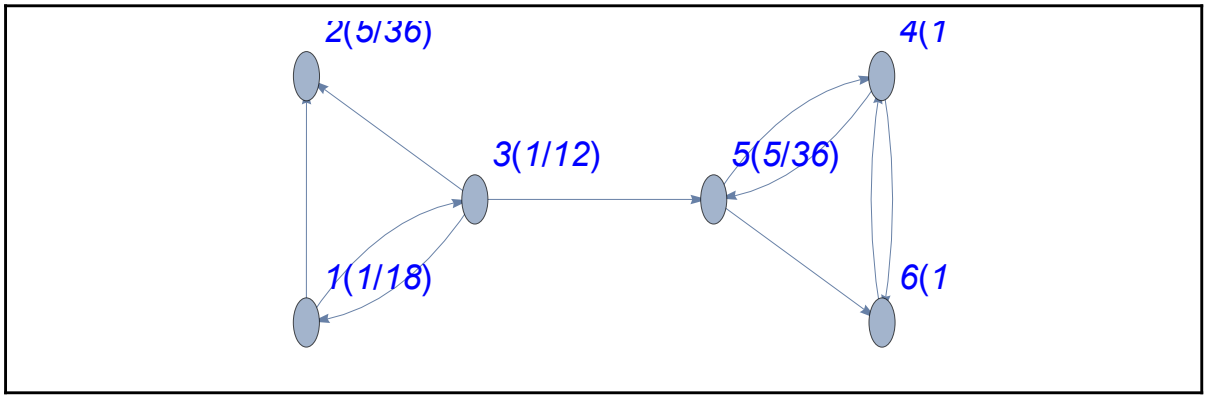
Faire le calcul des premières itérations sur le graphe ci-dessous.

```
g = Graph[{1, 2, 3, 4, 5, 6},
  {1 → 2, 1 → 3, 3 → 1, 3 → 2, 3 → 5, 4 → 5, 4 → 6, 5 → 4, 5 → 6, 6 → 4},
  VertexLabels → {1 → "1", 2 → "2", 3 → "3", 4 → "4", 5 → "5", 6 -> "6"},
  VertexLabelStyle → Directive[Blue, Italic, 18], VertexSize → Medium]
```

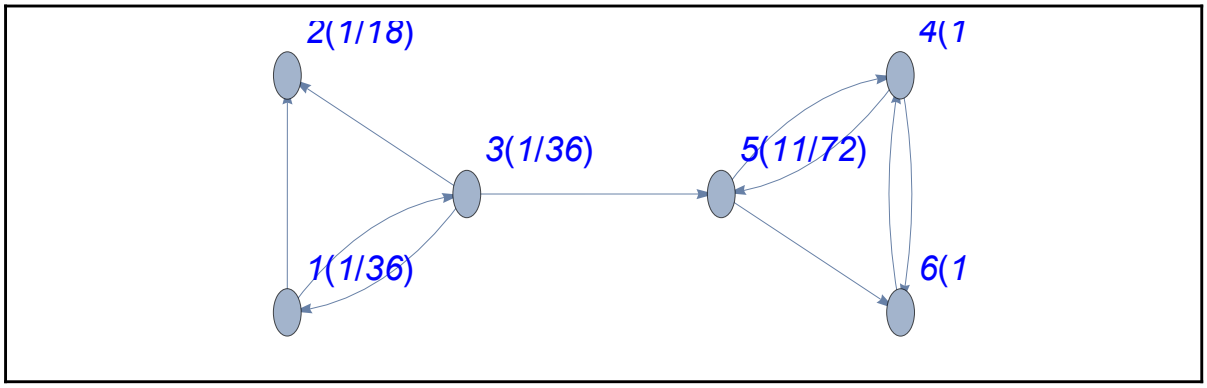


Deux itérations

```
g1it1 = Graph[{1, 2, 3, 4, 5, 6},
  {1 → 2, 1 → 3, 3 → 1, 3 → 2, 3 → 5, 4 → 5, 4 → 6, 5 → 4, 5 → 6, 6 → 4},
  VertexLabels → {1 → "1(1/18)", 2 → "2(5/36)",
    3 → "3(1/12)", 4 → "4(1/4)", 5 → "5(5/36)", 6 -> "6(1/6)"},
  VertexLabelStyle → Directive[Blue, Italic, 18], VertexSize → Medium]
```



```
g1it2 = Graph[{1, 2, 3, 4, 5, 6},
  {1 → 2, 1 → 3, 3 → 1, 3 → 2, 3 → 5, 4 → 5, 4 → 6, 5 → 4, 5 → 6, 6 → 4},
  VertexLabels → {1 → "1(1/36)", 2 → "2(1/18)", 3 → "3(1/36)",
    4 → "4(17/72)", 5 → "5(11/72)", 6 -> "6(14/72)"},
  VertexLabelStyle → Directive[Blue, Italic, 18], VertexSize → Medium]
```



Exprimer l'algorithme itératif sous forme matricielle.

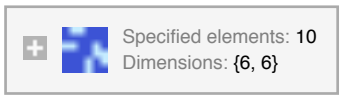
Matrice d'adjacence et marche aléatoire.

```
outDegree[g_?DirectedGraphQ] :=
 Total[AdjacencyMatrix[g], {2}]
```

```
outDegree[g]
```

```
{2, 0, 3, 2, 2, 1}
```

```
AdjacencyMatrix[g]
```

```
SparseArray[  ]
```



```
MatrixForm[%]
```

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

```
Total[%%, {2}]
```

```
{2, 0, 3, 2, 2, 1}
```

```
addEdgeWeight[g_?DirectedGraphQ] :=
 Module[{outDegrees = outDegree[g],
   edges = EdgeList[g], h = g},
  Do[PropertyValue[{h, edges[[i]]},
     EdgeWeight] =
    1 / outDegrees[[edges[[i]][[1]]]],
   {i, Length[edges]}
  ];
  h
 ]
```

```
EdgeList[g]
```

$$\{1 \leftrightarrow 2,\ 1 \leftrightarrow 3,\ 3 \leftrightarrow 1,\ 3 \leftrightarrow 2,\ 3 \leftrightarrow 5,\ 4 \leftrightarrow 5,\ 4 \leftrightarrow 6,\ 5 \leftrightarrow 4,\ 5 \leftrightarrow 6,\ 6 \leftrightarrow 4\}$$

```
EdgeList[g][[6]][[1]]
```

$$4$$

```
outDegree[g][[%]]
```

$$2$$

```
g = addEdgeWeight[g];
H = WeightedAdjacencyMatrix[g];
MatrixForm[%]
```

$$\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Cette matrice modélise un processus de marche aléatoire sur le graphe, chaque arête étant équiprobable.

Algorithme itératif.

$$p^{(k+1)\intercal} = p^{k\intercal} H$$

```
e = Table[1 / Length[H], {Length[H]}]
```

$$\left\{\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right\}$$

```
e.H
```

$$\left\{\frac{1}{18}, \frac{5}{36}, \frac{1}{12}, \frac{1}{4}, \frac{5}{36}, \frac{1}{6}\right\}$$

```
(e.H).H
```

$$\left\{\frac{1}{36}, \frac{1}{18}, \frac{1}{36}, \frac{17}{72}, \frac{11}{72}, \frac{7}{36}\right\}$$

```
pr0[h_, k_] :=
 Module[
  {e = Table[1 / Length[h], {Length[h]}],
   r = e},
  Do[r = r.H, {k}];
  r
 ]
pr0[H, 10]
```

$$\left\{\frac{1}{46\,656}, \frac{1}{23\,328}, \frac{1}{46\,656}, \frac{397\,901}{1\,492\,992}, \frac{199\,283}{1\,492\,992}, \frac{9331}{46\,656}\right\}$$

```
N[%]
```

```
{0.0000214335, 0.0000428669, 0.0000214335,
 0.266512, 0.133479, 0.199996}
```

Discuter la complexité de cet algorithme.

Chaque itération implique la multiplication d'un vecteur par une matrice. Donc la complexité théorique de chaque itération pourrait être $O(n^2)$. Mais la matrice H est creuse.

Or la complexité de la multiplication d'un vecteur par une matrice creuse est bornée en proportion linéaire au nombre de valeurs non nulles de la matrice. La complexité de chaque itération de l'algorithme est donc $O(\text{nnz}(H))$. Or le nombre moyen de liens sortants pour une page web est d'environ 10. Donc la complexité de chaque itération est $O(n)$.

Représenter une matrice creuse d'une manière efficace en terme de mémoire.

Le principe est de ne pas stocker les zéro. Il y a deux approches courantes : CRS (Compressed Row Storage) et CCS (Compressed Column Storage). Par exemple, pour CRS, les valeurs non nulles sont stockées dans un bloc mémoire contigu dans leur ordre d'apparition lorsque la matrice est lue ligne par ligne. Par exemple, pour la matrice *H* :

```
val = {H[[1, 2]], H[[1, 3]], H[[3, 1]],
  H[[3, 2]], H[[3, 5]], H[[4, 5]], H[[4, 6]],
  H[[5, 4]], H[[5, 6]], H[[6, 4]]}
```

$$\left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 1\right\}$$

Un second tableau mémorise l'indice de la colonne pour chaque entrée du tableau val :

```
colInd = {2, 3, 1, 2, 5, 5, 6, 4, 6, 4}
```

```
{2, 3, 1, 2, 5, 5, 6, 4, 6, 4}
```

Un troisième et dernier tableau mémorise les positions dans val qui commencent une ligne.
Souvent, on posera que rowPtr[[n + 1]] = nnz + 1 afin que rowPtr[[i + 1]] – rowPtr[[i]] soit le nombre d'éléments non nuls de la ligne i.

```
rowPtr = {1, 3, 3, 6, 8, 10, 11}
```

```
{1, 3, 3, 6, 8, 10, 11}
```

Souvent, on représentera rowPtr en décrémentant chaque entrée de la valeur 1. Cela pour simplifier l'écriture des algorithmes qui manipulent des matrices creuses (nous allons le voir un peu plus tard sur l'exemple concret de l'algorithme de multiplication d'un vecteur par une

matrice). C'est par exemple la représentation adoptée par *Mathematica* (Matlab adopte par défaut une représentation CCS. Cleve Moler l'un des fondateurs de Matlab, était l'un des auteurs de la librairie Fortran LINPACK. Or, en Fortran, les matrices sont stockées en mémoire colonne par colonne) :

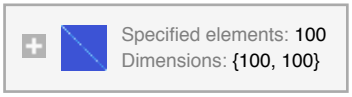
```
FullForm[SparseArray[H]]
```

```
SparseArray[Automatic, List[6, 6], 0,
 List[1, List[List[0, 2, 2, 5, 7, 9, 10],
   List[List[2], List[3], List[1],
    List[2], List[5], List[5], List[6],
    List[4], List[6], List[4]]],
  List[Rational[1, 2], Rational[1, 2],
   Rational[1, 3], Rational[1, 3],
   Rational[1, 3], Rational[1, 2],
   Rational[1, 2], Rational[1, 2],
   Rational[1, 2], 1]]]
```

Un petit exemple pour se rendre compte de l'intérêt (du point de vue de l'espace pour l'instant) d'une représentation adaptée d'une matrice creuse :

```
d = RandomReal[1, {100}];
```

```
sp2 = DiagonalMatrix[SparseArray[d]]
```

SparseArray[]


```
{ByteCount[sp2],
 ByteCount[DiagonalMatrix[d]]}
```

```
{3160, 80 152}
```

Ecrire de manière efficace la multiplication d'une matrice creuse par un vecteur.

Cas d'une matrice pleine

Dans le cas d'une matrice pleine il y a autant de multiplications à faire que d'entrées dans la matrice (i.e. n^2).

```
mulDense[m_, nbRows_, nbCols_, x_] :=
 Module[{y = Table[0, {nbCols}]},
  Do[
   y[[j]] += x[[i]] * m[[i, j]],
   {i, 1, nbRows},
   {j, 1, nbCols}
  ];
  y]
```

```
mulDense[H, Length[H], Length[H[[1]]], e]
```

$$\left\{\frac{1}{18}, \frac{5}{36}, \frac{1}{12}, \frac{1}{4}, \frac{5}{36}, \frac{1}{6}\right\}$$

Cas d'une matrice creuse représentée dans le format CRS

Dans le cas d'une matrice creuse il y a autant de multiplica tions à faire que d'entrées non nulles dans la matrice.

```
mulSparse[val_, colInd_,
  rowPtr_, nbRows_, nbCols_, x_] :=
 Module[{y = Table[0, {nbCols}],
   curValIndex = 1},
  Do[
   curValIndex = (rowPtr[[i]] - 1) + j;
   y[[colInd[[curValIndex]]]] +=
    x[[i]] * val[[curValIndex]],
   {i, 1, nbRows},
   {j, 1, (rowPtr[[i + 1]] - rowPtr[[i]])}
  ];
  y]
```

En stockant dans rowPtr {0, 2, 2, 5, 7, 9, 10} au lieu de {1, 3, 3, 6, 8, 10, 11} on aurait curValIndex = rowPtr[[*i*]] + *j*, ce qui économise une soustraction à chaque itération.

```
mulSparse[val, colInd, rowPtr, 6, 6, e]
```

$$\left\{\frac{1}{18}, \frac{5}{36}, \frac{1}{12}, \frac{1}{4}, \frac{5}{36}, \frac{1}{6}\right\}$$

Se poser des questions sur l'algorithme itératif dans son état actuel.

- L'algorithme converge-t-il ?
- La convergence dépend-elle du vecteur initial $p^{(0)\,T}$?
- S'il converge, à quelle vitesse converge-t-il ? (i.e. combien d'itérations avant d'avoir une bonne estimation du point de convergence ?)
- Les résultats obtenus sur la matrice *H* sont-ils pertinents ?
- Que se passe-t-il en présence de cycles (voir graphe ci-dessous) ?

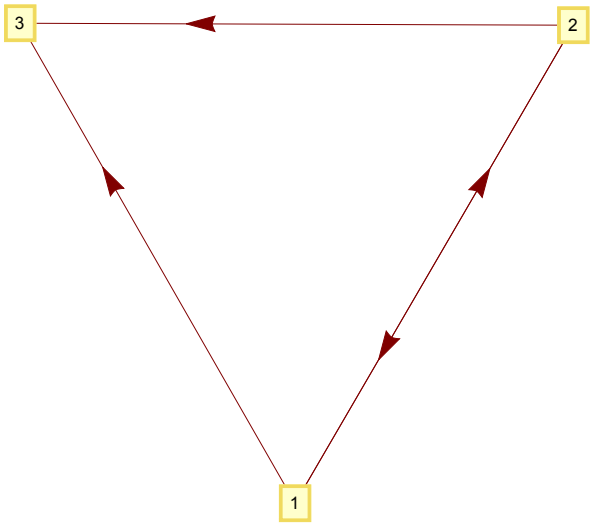
```
gCycle = Graph[{1 → 2, 2 → 1}];

GraphPlot[gCycle, VertexLabeling → True, DirectedEdges → True]
```

- Que se passe-t-il en présence de noeuds sans liens sortants ?

```
gSink = Graph[{1 → 2, 1 → 3, 2 → 1, 2 → 3}];

GraphPlot[gSink, VertexLabeling → True, DirectedEdges → True]
```



Noeuds absorbants

Commençons par considérer le problème des boucles et des noeuds absorbants.
En fait, la manière dont Brin et Page ont résolu ces problèmes a permis d'assurer la convergence de l'algorithme. Ceci s'explique grâce à des résultats issus de la théorie des chaînes de Markov. Mais il n'était pas fait mention de cette dernière dans les travaux qui ont introduit l'algorithme du PageRank.

Les noeuds absorbants correspondent aux lignes nulle de la matrice de marche aléatoire.

```
gSink = addEdgeWeight[gSink];
```

```
WeightedAdjacencyMatrix[gSink] // MatrixForm
```

$$\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 \end{pmatrix}$$

Au fil des itérations, ces noeuds absorbent les scores de leurs voisins.
D'un point de vue matricielle, on dit que la matrice, que nous avons appelée jusqu'ici abusivement "matrice de marche aléatoire", n'est pas stochastique car toutes ses lignes ne correspondent pas à des probabilités, i.e. la somme de certaines lignes n'est pas 1.
Le vecteur binaire qui identifie les noeuds absorbants est souvent nommé "dangling node vector".
Nous le noterons *a*.

```
danglingVector[m_] := Module[{a},
  (* m is the adjacency matrix of the
   graph before taking care of the
     dangling nodes.
     returns a binary vector
   identifying the dangling nodes.*)
  a = Table[0, {Length[m]}];
  Do[a[[i]] = If[Total[m[[i]]] == 0, 1, 0],
   {i, Length[m]}
  ];
  a
 ]
danglingVector[
 WeightedAdjacencyMatrix[gSink]]
{0, 0, 1}
```

Proposer une solution pour rendre la matrice *H* stochastique

Pour rendre la matrice *H* stochastique, on lui ajoute la matrice de rang 1 (car c'est un produit tensoriel de deux vecteurs) : $a\left(\frac{1}{n}\, e^{\intercal}\right)$.

$$S = H + a\left(\frac{1}{n}\, e^{\intercal}\right)$$

```
Outer[Times,
  danglingVector[AdjacencyMatrix[gSink]],
  1 / 3 * Table[1, {3}]] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

```
makeStochastic[mat_] :=
 Module[{m = mat, n = Length[mat]},
  (* m is the adjacency matrix of the
   graph before taking care of the
     dangling nodes.
     returns an adjacency matrix
   where the lines corresponding
     to dangling nodes have been
   replaced by nodes that have an
     equal probability to jump
   to any other node.*)
  m = m + Outer[Times, danglingVector[m],
     1 / n * Table[1, {n}]]
 ]
```

```
makeStochastic[WeightedAdjacencyMatrix[
    gSink]] // MatrixForm
```

$$\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

```
S = makeStochastic[H];
```

```
S // MatrixForm
```

$$\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Quelques éléments à propos de la théorie des chaînes de Markov.

Une matrice stochastique représente une chaîne de Markov, autrement dit un processus stochastique discret : un processus qui est une suite discrète de transitions aléatoires d'un état vers un autre. Si P est la matrice de transition d'une chaîne de Markov, P_{ij} est la probabilité de passer de l'état i à l'état j.

Une matrice stochastique possède un vecteur propre à gauche de valeur propre 1 :
$\pi^T P = \pi^T$. Ce vecteur propre principal π correspond donc à une distribution de probabilité stable pour la transformation modélisée par la chaîne de Markov.

Si la matrice stochastique P est irréductible (i.e. après suffisamment d'itération, il existe toujours un chemin entre deux noeuds quelconques du graphe) et apériodique (un état i est de période k si, dans l'état i, tous les passages futurs par i auront lieu après un nombre de transitions multiple de k ; une matrice est apériodique si tous ses états sont apériodiques ; si un état d'une matrice irréductible est apériodique alors tous ses états sont apériodiques) alors le processus itératif $\pi^{(k+1)^T} = \pi^{(k)^T} P$ converge vers un vecteur unique et positif, le vecteur propre principal π.

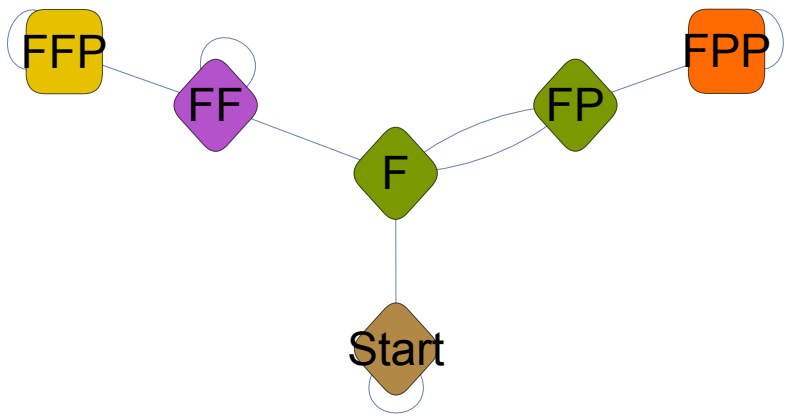
Une chaîne de Markov irréductible et apériodique est dite ergodique.

Exemple jouet d'application des chaînes de Markov

On lance une pièce non truquée et on veut mesurer en combien de coups en moyenne les configurations FFP (i.e. Face-Face-Pile) et FPP apparaissent. Aussi, quel est l'état final le plus probable ? Modéliser le problème sous la forme d'une chaîne de Markov.

Eléments de solution

```
markovProc = DiscreteMarkovProcess[1,
    {{1/2, 1/2, 0, 0, 0, 0}, {0, 0, 1/2,
      1/2, 0, 0}, {0, 0, 1/2, 0, 1/2, 0},
     {0, 1/2, 0, 0, 0, 1/2},
     {0, 0, 0, 0, 1, 0}, {0, 0, 0, 0, 0, 1}}];
Graph[{"Start", "F", "FF",
  "FP", "FFP", "FPP"}, markovProc,
 ImageSize → Medium, VertexLabelStyle ->
  Directive[Black, Plain, 24]]
```



Distribution stationnaire de cette chaîne de Markov :

```
StationaryDistribution[markovProc]
```

```
ProbabilityDistribution[
  2/3 Boole[x == 5] + 1/3 Boole[x == 6], {x, 1, 6, 1}]
```

La probabilité de l'état FFP est $\frac{2}{3}$, celle de l'état FPP est $\frac{1}{3}$.

Nombre moyen de lancés pour rejoindre l'état FFP :

```
Mean[FirstPassageTimeDistribution[
    markovProc, 5]] // N
```

```
5.66667
```

Nombre moyen de lancés pour rejoindre l'état FPP :

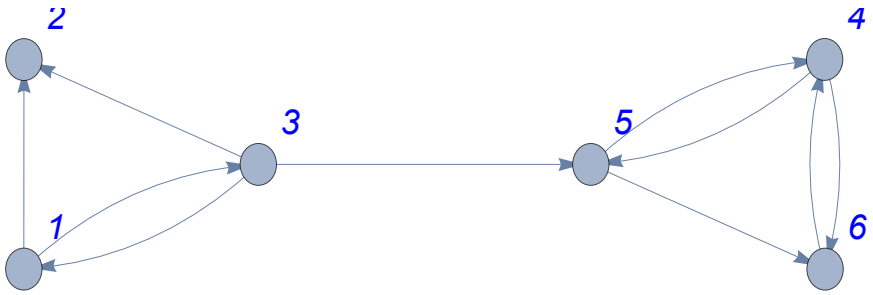
```
Mean[FirstPassageTimeDistribution[
    markovProc, 6]] // N
```

```
4.66667
```

Quelques éléments sur les puissances d'une matrice d'adjacence.

Rappelons le graphe *g* et sa matrice d'adjacence :

```
g
```




```
AdjacencyMatrix[g] // MatrixForm
```

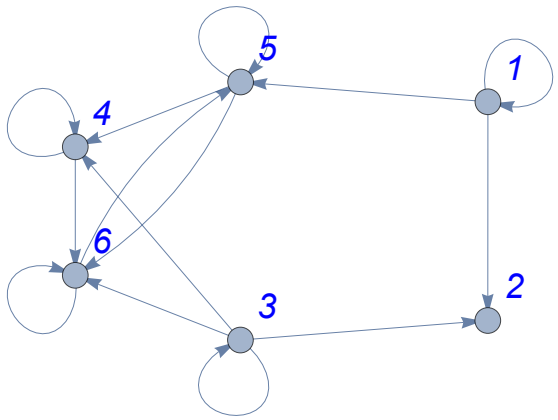
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

A quoi correspond g^2 ?

```
MatrixPower[AdjacencyMatrix[g], 2] //
 MatrixForm
```

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

```
AdjacencyGraph[{{1, 1, 0, 0, 1, 0}, {0, 0, 0, 0, 0, 0},
  {0, 1, 1, 1, 0, 1}, {0, 0, 0, 1, 0, 1}, {0, 0, 0, 1, 1, 1}, {0, 0, 0, 0, 1, 1}},
 VertexLabels → {1 → "1", 2 → "2", 3 → "3", 4 → "4", 5 → "5", 6 -> "6"},
 VertexLabelStyle → Directive[Blue, Italic, 18], VertexSize → Medium]
```



Une arête relie deux noeuds s'il existe au moins un chemin de taille 2 entre eux. Le poids de l'arête correspond au nombre de ces chemins.

A quoi correspond g^3 ? A quoi correspondent géométriquement les éléments sur la diagonale ?

```
MatrixPower[AdjacencyMatrix[g], 3] //
 MatrixForm
```

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 & 2 \\ 0 & 0 & 0 & 2 & 1 & 2 \\ 0 & 0 & 0 & 2 & 0 & 1 \end{pmatrix}$$

Que pourrions-nous définir comme étant le diamètre d'un graphe ?

La taille du plus grand plus petit chemin entre deux noeuds du graphe, pour tous les couples de noeuds possibles. Nous pouvons par exemple sommer les puissances successives de la matrice d'adjacence jusqu'à obtenir un graphe complet (une matrice sans 0).

A quoi correspondent les puissances d'une matrice stochastique ?

```
MatrixPower[S, 1] // MatrixForm
```

$$\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

```
MatrixPower[S, 2] // MatrixForm
```

$$\begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{12} & \frac{1}{12} & \frac{1}{4} & \frac{1}{12} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{9} & \frac{5}{18} & \frac{1}{6} & \frac{7}{36} \\ \frac{1}{18} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{1}{18} & \frac{2}{9} \\ 0 & 0 & 0 & \frac{3}{4} & 0 & \frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

```
MatrixPower[S, 10] // N // MatrixForm
```

```
0.00711303  0.0122132  0.00811341  0.432478
0.00407105  0.00708434  0.00475434  0.436524
0.00540894  0.00950869  0.00642974  0.430499
        0.          0.          0.  0.448242
        0.          0.          0.  0.447266
        0.          0.          0.    0.4375
```

Peut-on maintenant comprendre l'importance de l'ergodicité d'une matrice ?

Rendre S ergodique.

L'idée de Brin et Page pour permettre la découverte d'une distribution stable par multiplication itérée de la matrice stochastique est d'introduire la métaphore du "surfeur aléatoire" (random surfer). Cette métaphore permet en fait de rendre la matrice ergodique. Le surfeur suit la structure des liens du web, sauf que de temps en temps il décide d'entrer une URL dans la barre de navigation afin de sauter vers une page quelconque non nécessairement connectée à la page courante. On appelle cette opération la téléportation. La téléportation est aléatoire : chaque page est équiprobable en tant que destination d'une opération de téléportation. On fait une combinaison convexe de *S* et de la matrice de rang 1 qui représente l'opération de téléportation :

$$G = \alpha S + (1 - \alpha) \frac{1}{n} e e^{T}$$

```
makeErgodic[mat_, α_] :=
 Module[{m = mat, n = Length[mat], e},
  (* m is a stochastic matrix,
     i.e. the dangling
   nodes have been taken care of.
     returns an ergodic matrix
   obtained thanks to the
   "random surfer" method. *)
  e = Table[1, {n}];
  m = α m + (1 - α) (1 / n) Outer[Times, e, e]
 ]

G = makeErgodic[S, 0.85];
```

```
G // MatrixForm
```

$$\begin{pmatrix} 0.025 & 0.45 & 0.45 & 0.025 & 0.02 \\ 0.166667 & 0.166667 & 0.166667 & 0.166667 & 0.166 \\ 0.308333 & 0.308333 & 0.025 & 0.025 & 0.308 \\ 0.025 & 0.025 & 0.025 & 0.025 & 0.4 \\ 0.025 & 0.025 & 0.025 & 0.45 & 0.02 \\ 0.025 & 0.025 & 0.025 & 0.875 & 0.02 \end{pmatrix}$$

Maintenant que la matrice est ergodique, la méthode de la puissance itérée nous donne un résultat un peu plus satisfaisant :

```
MatrixPower[G, 10] // N // MatrixForm
```

$$\begin{pmatrix} 0.0529125 & 0.0757492 & 0.0587855 & 0.346738 & 0.199008 & 0.266807 \\ 0.0523136 & 0.0747394 & 0.0581242 & 0.347534 & 0.199555 & 0.267734 \\ 0.052577 & 0.0752167 & 0.058454 & 0.346348 & 0.199943 & 0.267461 \\ 0.0515121 & 0.0733447 & 0.0571882 & 0.349841 & 0.19931 & 0.268804 \\ 0.0515121 & 0.0733447 & 0.0571882 & 0.349649 & 0.199503 & 0.268804 \\ 0.0515121 & 0.0733447 & 0.0571882 & 0.347726 & 0.201233 & 0.268996 \end{pmatrix}$$

Bénéficier du fait que H est creuse

La matrice G, ergodique, est dense. Donc le calcul de la distribution stationnaire par la méthode de la puissance itérée ne pourra plus bénéficier des optimisations propres aux matrices creuses. Exprimer le calcul $\pi^{(k+1)^\intercal} = \pi^{(k)^\intercal} G$ afin de pouvoir bénéficier du fait que la matrice H est creuse.

Elements de réponse

$$\begin{aligned} G &= \alpha S + (1-\alpha)\,\frac{1}{n}\,e\,e^\intercal \\ &= \alpha\left(H + \frac{1}{n}\,a\,e^\intercal\right) + (1-\alpha)\,\frac{1}{n}\,e\,e^\intercal \\ &= \alpha H + (\alpha a + (1-\alpha)\,e)\,\frac{1}{n}\,e^\intercal \end{aligned}$$

Donc, nous avons :

$$\pi^{(k+1)^\intercal} = \alpha\,\pi^{(k)^\intercal} H + \left(\alpha\,\pi^{(k)^\intercal} a + (1-\alpha)\,\pi^{(k)^\intercal} e\right)\frac{1}{n}\,e^\intercal$$

Ainsi, une itération est de complexité $O(n)$.

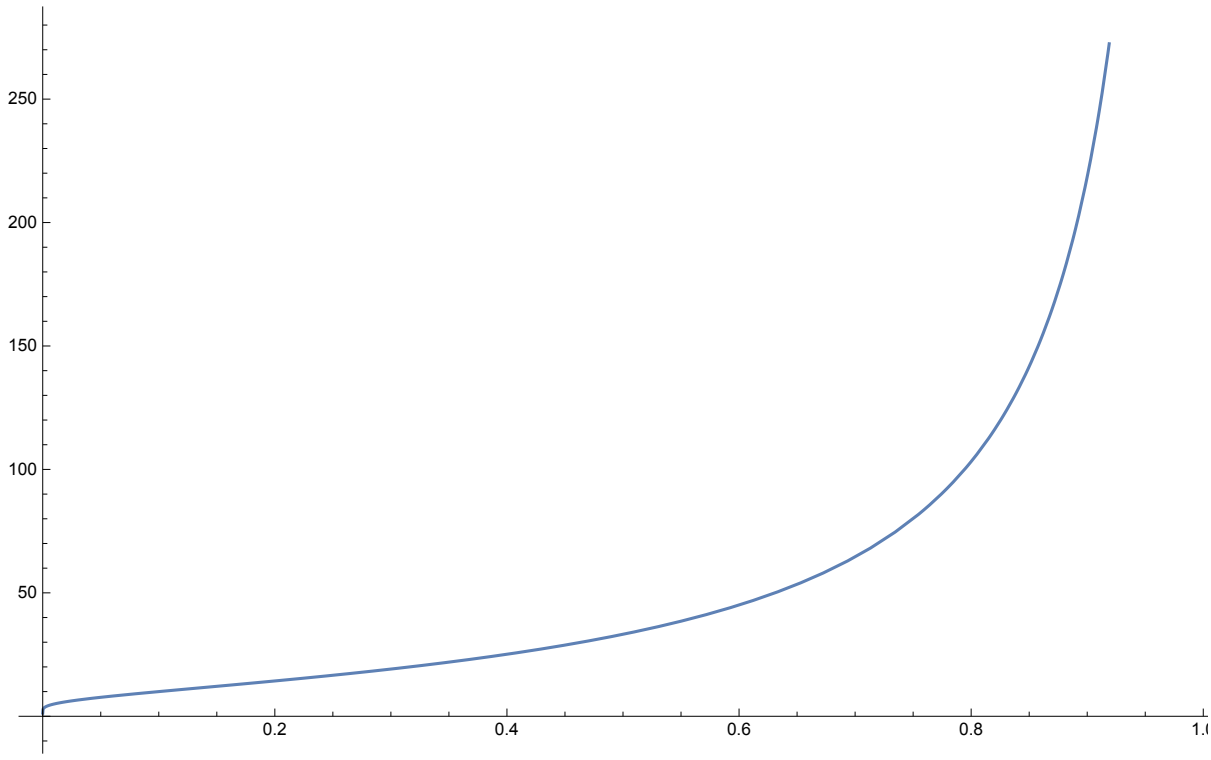
Vitesse de convergence

On peut montrer que la vitesse de convergence du PageRank est asymptotiquement égale à celle de la convergence vers 0 de α^n. Brin et Page ont initialement choisi une valeur $\alpha = 0.85$. Ils arrêtaient l'algorithme après une cinquantaine d'itérations. Est-ce raisonnable ?

Combien d'itérations sont-elle nécessaires pour une précision de $10^{-\tau}$?

$$\alpha^n \leq 10^{-\tau}$$
$$n \log_{10}(\alpha) \leq -\tau$$
$$n \geq -\tau \,/\, \log_{10}(\alpha)$$

```
Plot[-10 Log[10] / Log[α], {α, 0, 1}]
```



Spam

Soit un Web de N pages. Un spammer ajoute la page x. Il possède par ailleurs un ensemble de k pages. Comment doit-il structurer ses pages pour qu'ajoutées aux N pages, le pagerank de x soit optimal ?