INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

**Projet Scientifique Encadré**
**STPI/P6/2024**

# L'algorithme PageRank ou le succès du moteur de recherche de Google


**Étudiants :**
Emmanuel CHARRIER
Gauvain NOITON
Esteban AYORA-GUIA
Paul FOUGERAY
Paul KERFANT
Bilal BAMMOU

**Enseignant responsable du projet :**
Alexis LECOMTE

**Date de remise du rapport :** 15/06/2024

**Référence du projet :** Projet numéro 5

**Intitulé du projet :** L'algorithme PageRank ou le succès du moteur de recherche de Google

**Type de projet :** model

**Objectifs du projet :** Analyser et reproduire l'algorithme PageRank du moteur de rechercher Google, qui permet de classer les sites par pertinence lors d'une recherche

**Mots-clefs du projet :** PageRank, Informatique,

**Remerciements :** Nous remercions notre enseignant responsable du projet monsieur LECOMTE Alexis pour nous avoir accompagné tout au long de ce travail.

# Table des matières

# Introduction

- Dans un monde où Google est incontestablement le moteur de recherche le plus utilisé au monde, avec environ 8.5 milliards de recherches quotidiennes en moyenne, peu de personnes savent de quelle façon le géant nous propose les sites les plus pertinents. Lorsque nous parlons de recherche, nous "parlons" de taper directement dans le moteur de recherche, ce qui est différent du fait d'accéder directement à un site. Pour illustrer cela : une recherche serait d'écrire 'Insa Rouen' dans la barre de recherche de Google, tandis que taper l'URL `https://moodle.insa-rouen.fr` ne compte pas comme une recherche, idem pour les raccourcis.

- L'objectif de ce projet est de proposer et de comprendre le fonctionnement théorique de cet algorithme, notamment en comprenant les différents facteurs qui influent sur le "classement" des sites effectués par Google, pour ensuite implémenter cet algorithme dans un langage de notre choix. Dans notre cas nous avons choisi Python pour son efficacité (notamment les nombreuses bibliothèques disponibles) et notre familiarité avec ce langage.

- L'algorithme PageRank, conçu par Larry Page et Sergey Brin, s'appuie principalement sur l'analyse des hyperliens sur le web. Chaque lien agit comme un vote en faveur de la pertinence de la page à laquelle il est lié. En effet, sur une page du web, on peut retrouver des hyperliens, faisant référence à d'autres pages. Le score d'une page est d'autant plus élevé qu'elle est citée par des pages au score élevé. Nous nous sommes concentrés sur cet aspect du PageRank, cependant, d'autres facteurs rentrent en compte, comme le nombre d'occurrences de mots-clés sur les pages, etc...

# Chapitre 1

# Méthodologie, organisation du travail

- Notre première mesure a été de faire des recherches individuelles sur le PageRank, afin de se familiariser et de comprendre notre sujet, pour ensuite mettre en commun nos découvertes et être au même niveau de connaissance. Nous nous sommes très vite aperçus que tout cela tournait autour des hyperliens.
  Nous avons donc pris la décision de nous concentrer sur ce premier critère, et de nous répartir en 3 groupes de 2 :

  1. Le premier groupe (celui de Paul K. et Gauvain) s'est concentré sur les mathématiques derrière l'algorithme, et sur la façon dont les interactions entre les sites sont représentées (nous expliquerons ce point très important par la suite).
  2. Ensuite le second groupe (Emmanuel et Paul F.) a réalisé un "crawler", un programme ayant pour but de récupérer les liens sortants d'un site, afin de créer une mini-toile nécessaire pour modéliser le PageRank.
  3. Enfin le dernier groupe (Bilal et Esteban) a fait le lien entre le modèle mathématique du premier groupe et le langage informatique permettant de modéliser au mieux le problème.

  L'échange entre nos groupes a été primordial afin d'avancer le plus efficacement possible.

- Organigramme des tâches réalisées et des étudiants concernés :

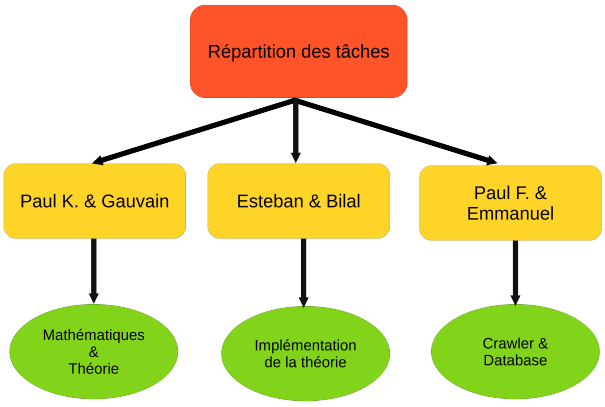


FIGURE 1 – Organigramme

- Lors de notre travail, il y a eu évidemment des échanges entre la partie théorique et son implémentation mais aussi, entre l'implémentation de la partie théorique et la récupération de données, afin d'avoir des programmes compatibles utilisant les mêmes conventions et structures de données.

- Une fois les travaux terminés, nous nous sommes réunis afin de mettre en commun, le crawler (outil permettant de récupérer les informations d'un site web, comme les pages qu'il référence avec les hyperliens, les mots présents sur le site...) et l'implémentation du PageRank (utilisant les hyperliens). Nous obtenons suite à nos exécutions des résultats concluants (qui seront développés dans le second chapitre). Nous devions maintenant trouver de nouvelles tâches à réaliser, notamment pour la présentation du projet.

- Nous nous sommes alors réunis de nouveau afin de trouver un moyen de présenter le plus efficacement le PageRank au jury. Suite à ça, nous avons décidé de réaliser plusieurs pages web afin d'illustrer la notion d'hyperliens ainsi qu'une animation permettant d'expliquer le surfeur aléatoire qui est une modélisation mathématique du comportement de l'internaute, permettant aussi d'expliquer la convergence du graphe du PageRank. Gauvain s'est occupé de développer les pages web et Paul K. s'est chargé de l'animation.

- Les autres membres du groupe se sont alors répartis sur les tâches qu'il nous restait, Bilal et Esteban se sont occupés du rapport et Emmanuel et Paul F. ont exploré la partie du PageRank (influences d'autres paramètres que les hyperliens)

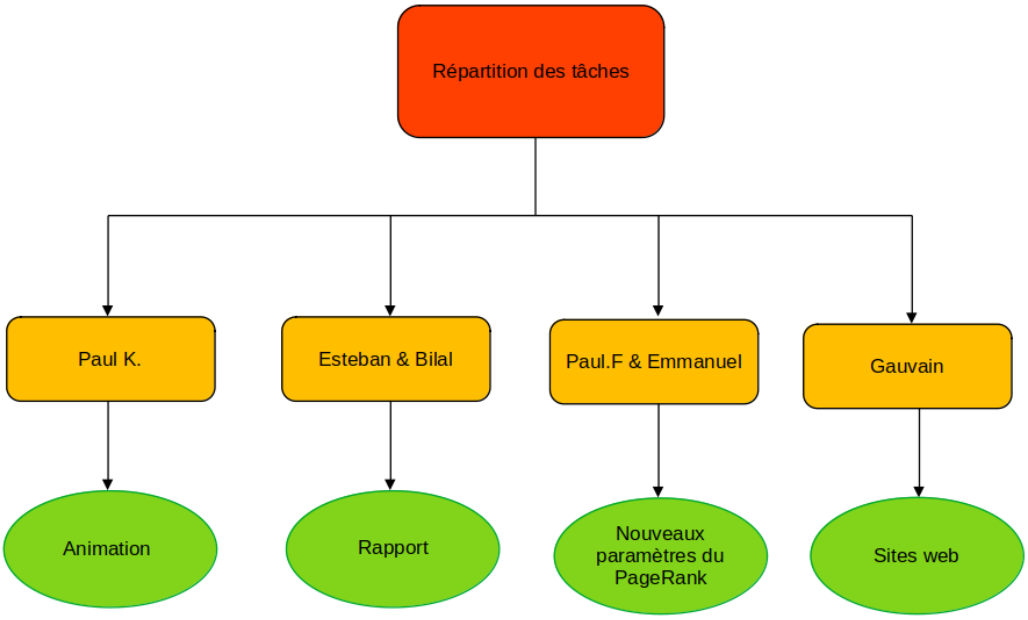


FIGURE 2 – Organigramme

# Chapitre 2

# Travail réalisé et résultats

## 2.1 Travail théorique

### 2.1.1 Introduction

Le but du projet est de comprendre le fonctionnement derrière l'algorithme ayant fait le succès de Google, le PageRank. L'informatique repose sur des notions mathématiques pour modéliser des solutions, et le PageRank ne déroge pas à cela.

Pour cela nous avons étudié l'aspect théorique du PageRank, puis nous avons reproduit un modèle de celui-ci.

### 2.1.2 Première notion théorique

L'algorithme PageRank s'appuie principalement sur l'analyse des hyperliens sur le web. Chaque lien agit comme un vote en faveur de la pertinence de la page à laquelle il est lié.

En effet, sur une page du web, on peut retrouver des hyperliens, faisant référence à d'autres pages. Ces liens, présents sur une page A, vont alors transmettre une partie du PageRank de la page A aux pages qu'ils réfèrent. Ainsi, plus le PageRank de la page A est grand, plus le PageRank transmis sera important.

Le score PageRank est déterminé en passant par un modèle : le surfeur aléatoire. Il correspond à un utilisateur qui se déplacerait sur le web en cliquant aléatoirement sur les liens de la page où il se trouve. Le surfeur aléatoire va parcourir Internet, étape par étape. Il ne peut se trouver que sur une seule page à la fois, et à chaque étape, il va sur une nouvelle page en cliquant sur un lien. Nous allons chercher à déterminer la probabilité que ce surfeur se trouve sur une page à l'étape $i$, connaissant cette probabilité pour l'étape $i-1$. Au bout d'un certain nombre d'étapes, ces probabilités se stabilisent, et c'est cette valeur stable (limite) que nous retenons comme PageRank final. Cette stabilisation progressive, appelée convergence, peut être démontrée grâce aux chaînes de Markov, un concept mathématique développé en annexes A et B.

Nous avons cependant un problème : les "Spider Traps" ou pièges. Ils correspondent à un petit groupe de sites dont les hyperliens ne font référence qu'à d'autres sites de ce groupe. Une fois dans un Spider Trap, le surfeur aléatoire ne peut plus en sortir. En utilisant simplement la méthode décrite plus haut, les PageRank des sites hors du piège convergent vers $0$, ce qui n'est pas réaliste ou souhaitable. C'est pourquoi on utilise un "facteur d'amortissement" (Damping Factor) pour remédier à ce problème.

Le Damping Factor $\beta$ représente la probabilité que le surfeur aléatoire continue son exploration du web en cliquant sur un lien du site sur lequel il se trouve. Autrement dit, $1 - \beta$ représente la probabilité que l'utilisateur se téléporte vers un site totalement aléatoire. Nous pouvons voir ça comme le fait que l'utilisateur retourne sur son moteur de recherche ou une liste de sites (favoris, marque-pages...), pour y choisir un site aléatoire. Tous les sites ont une probabilité égale d'être choisis par cette méthode.

Cette méthode permet d'éliminer les Spider Traps, en rajoutant artificiellement des liens entre tous les sites. Cependant, cette méthode correspond aussi à un véritable comportement de l'utilisateur, et n'est donc pas irréaliste ou trop artificielle.

La valeur souvent choisie pour le facteur d'amortissement est de 0.85. Google lui-même utiliserait cette valeur, bien que cela ne soit pas confirmé car l'entreprise reste très secrète sur son algorithme. Cette valeur a sûrement été déterminée par une étude empirique du comportement des utilisateurs, bien qu'elle puisse se justifier par d'autres manières : une valeur trop proche de 0 donne lieu à des téléportations très fréquentes, et donc à des scores PageRank uniformes, alors qu'une valeur proche de 1 ralentit considérablement la convergence du PageRank, augmentant ainsi le temps de calcul. 0.85 offre donc un bon compromis.

### 2.1.3 Premier problème

Certains utilisateurs pourraient être tentés de gonfler artificiellement leur note de PageRank en référençant de manière excessive leur page avec plusieurs fausses pages, ou en achetant des liens. Il y a aussi la technique de l'auto-référencement. Les moteurs de recherche combattent cela en employant des algorithmes d'analyse de liens pour détecter et pénaliser un tel comportement.

Il se pose ensuite la question des pages n'ayant aucun lien entrant (c'est à dire aucun site ne réfère cette page et n'utilise son hyperlien). Ces pages, appelées pages orphelines, reçoivent un PageRank faible ou nul.

### 2.1.4 Première formule et représentation matricielle

Initialisation de chaque site :

$$PR(x) = 100\%/N$$

pour N le nombre de noeuds dans le graphique.

$$PR(x) = \frac{1-\lambda}{N} + \lambda \sum_{x}^{y} (\frac{PR(y)}{out(y)})$$

avec $\lambda$ le facteur d'amortissement de 0.85 et out(y) le nombre de liens vers les sites externes à y.

Pour le reste de nos tests, nous représenterons les sites et les liens entre eux par un graphe orienté comme celui ci-dessous.

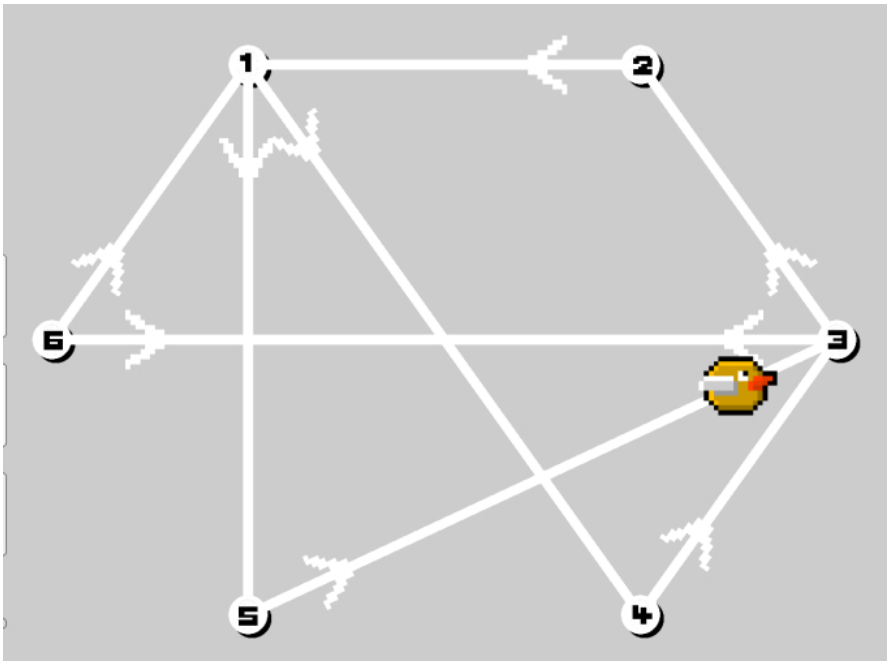


FIGURE 1 – Animation surfeur aléatoire

Ce graphique peut se représenter aussi sous forme de matrice. Dans cette matrice, chaque ligne correspond aux sites pointés par le site correspondant à la ligne :

- Par exemple le site 1 référence les sites 4 et 5.
- Le site 2 référence le site 1.
- Le site 3 référence les sites 2 et 6.
- Le site 4 référence le site 3.
- Le site 5 référence le site 3.
- Le site 6 référence les sites 1 et 3.

$$\begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \end{pmatrix} \tag{2.1}$$

Nous pouvons alors aussi représenter l'initialisation sous la forme d'une matrice où n, le nombre de sites, est égal à 6 :

$$\begin{bmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix} \tag{2.2}$$

Nous réalisons n fois le produit matriciel de (2.1) et (2.2) jusqu'à la convergence des résultats. Nous obtenons par ce produit matriciel une matrice d'une ligne et de 5 colonnes ayant pour valeurs les PageRank des sites (cf Annexe A : preuve de la convergence). 2 2

Voici le produit en prenant en considération le coefficient q correspondant au surfeur aléatoire fixé à 0.85. Les sommes des termes de chaque ligne sont toujours égales à 1.

$$\begin{bmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{q}{2}+\frac{1-q}{6} & \frac{q}{2}+\frac{1-q}{6} & \frac{1-q}{6} \\ q+\frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} \\ \frac{1-q}{6} & \frac{q}{2}+\frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{q}{2}+\frac{1-q}{6} \\ \frac{1-q}{6} & \frac{1-q}{6} & q+\frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} \\ \frac{1-q}{6} & \frac{1-q}{6} & q+\frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} \\ \frac{q}{2}+\frac{1-q}{6} & \frac{1-q}{6} & \frac{q}{2}+\frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} & \frac{1-q}{6} \end{bmatrix}^n$$

Application numérique : $\begin{bmatrix} 0.208 & 0.143 & 0.279 & 0.113 & 0.113 & 0.143 \end{bmatrix}$

Il est intéressant de remarquer que la colonne 2 (= site 2) , dans la matrice initiale, reçoit une "puissance" de $\frac{1}{2}$, idem pour la colonne 4 , et pourtant dans les valeurs de PageRank, le site 2 est à 0.143 et le site 4 est à 0.113. Cela se justifie par le fait que le site 3, ayant un PageRank plus grand que le site 1, référence le site 2 tandis que le site 1 référence le 4, donc ce dernier reçoit moins de "puissance".

## 2.2 Création de la base de données

### 2.2.1 Introduction

Pour calculer le PageRank, il est essentiel de considérer plusieurs facteurs, notamment savoir quels sites composent la "toile" et connaître les liens entre eux. La section suivante se concentre sur la récupération de ces données et sur la manière de les rendre utilisables pour calculer un PageRank.

### 2.2.2 Crawler

En informatique, il existe un type de logiciel appelé "crawler". Le rôle d'un crawler est de parcourir les sites web de manière autonome, en collectant sur chaque site un maximum d'informations telles que les liens entrants/sortants, les hyperliens, les mots présents sur la page, etc. Ce robot informatique est essentiel pour notre travail, car c'est lui qui va nous fournir l'ensemble des données nécessaires.

Le fonctionnement du crawler est assez simple : lorsque nous lui fournissons une adresse URL, le robot envoie une requête à la page. En réponse, il reçoit tout le code HTML (HyperText Markup Language) de la page, qui décrit la structure de la page avec toutes les informations qui sont affichées ou non. Ensuite, nous utilisons ce code HTML en le passant à travers un parseur, que nous appelons analyseur syntaxique en français. En effet, par défaut, lorsque nous récupérons le code de la page, notre crawler est incapable de le comprendre. Il apparaît comme une simple suite de caractères sans logique qui pourrait tout aussi bien être mise de manière aléatoire, sans aucun sens. Le parseur va attribuer des "étiquettes" à chaque mot pour expliquer sa fonction au programme.

L'étape précédente nous permet ensuite de filtrer le contenu et de conserver uniquement les liens présents sur la page. Nous ajoutons ensuite chacun de ces liens à la file d'attente

du crawler, qui va recommencer l'opération pour chacun d'entre eux. C'est grâce à ce processus que nous créons la "toile", où pour chaque lien, nous obtenons les liens qu'il référence.

Pour réaliser ce processus, nous avons besoin d'utiliser plusieurs bibliothèques Python, notamment BeautifulSoup, qui sert à créer le parseur en une étape.

En essayant notre code, nous nous sommes rendu compte d'un problème majeur : le parcours des liens prenait beaucoup de temps. En effet, chaque lien prenait entre une demi-seconde et deux secondes. Nous avons donc décidé de paralléliser le processus. Autrement dit, nous avons créé plusieurs crawlers qui partagent une queue commune et qui analysent les pages en parallèle. Cette solution n'est pas parfaite, car la vitesse d'exécution de notre programme dépend aussi beaucoup de la qualité de la connexion internet, mais elle nous permet d'accélérer grandement le processus.

### 2.2.3 Base de données

Nous avons maintenant les données nécessaires, mais encore faut-il pouvoir les utiliser et les stocker. En effet, lorsque le crawler a fini de récupérer l'ensemble de la toile créée, elle n'est pas stockée par défaut, il faut donc relancer le parcours de la toile à chaque fois. Nous avons donc fait le choix de créer une base de données SQLite.

SQLite est une bibliothèque logicielle qui offre un système de gestion de base de données relationnelle embarquée, autonome et sans serveur. Les données sont stockées dans un unique fichier, simplifiant leur transfert et leur sauvegarde. Compact et léger, SQLite garantit la fiabilité des opérations. Principalement utilisée dans les applications mobiles, les systèmes embarqués et les logiciels nécessitant une intégration rapide, SQLite peut cependant présenter des limitations pour les applications à forte charge et les bases de données volumineuses.

Après avoir créé la base de données, nous avons élaboré une méthode pour la remplir. Nous avons opté pour l'utilisation de l'URL d'un site en tant que clé principale pour chaque observation dans la base de données. À chaque URL est associée un tableau contenant l'ensemble des liens sortants du site correspondant. À présent, nous sommes prêts à mettre en place la méthode inverse, qui consiste à recréer une toile depuis la base de données. Celle-ci permettra à d'autres groupes de calculer le score de PageRank de chaque page.

## 2.3 Implémentation du PageRank

### 2.3.1 Introduction

Après avoir étudié la théorie autour du PageRank et avoir mis en place une base de données contenant les liens entrants et sortants des différents sites web pris en considération, il est maintenant nécessaire de programmer la méthode matricielle approchée du PageRank à partir de ces données. Le but est donc d'afficher les différents scores de chaque page.

### 2.3.2 Programation

Pour ce faire nous avons utilisé python, de par sa facilité et une certaine bibliothèque que nous utilisons déjà pour manipuler des matrices et des vecteurs : Numpy. En effet cette dernière librairie est un choix judicieux car cet outil permet d'effectuer des opérations plus

rapidement et efficacement que les listes de base de Python. Ces mêmes raisons s'appliquent aussi pour la compatibilité avec nos camarades ayant réalisé le crawler, qui pour les mêmes raisons ont opté pour Python.

À partir de la base de données nous créons tout d'abord notre toile composée des sites et des liens sortants. Nous utilisons 3 fonctions, la première, lire_lien, compte le nombre de liens sortants de chaque site. Ensuite les 2 autres fonctions servent à initialiser chacune des matrices de la même manière que dans la partie théorique. Il ne reste plus que le programme principal pour effectuer le calcul, pour cela nous utilisons une boucle qui effectue les produits matriciels, jusqu'à convergence de la matrice. Pour que la boucle s'arrête nous précisons une certaine précision de différence entre 2 itérations, ici c'est 0.0001.

## 2.4 Term Frequency Inververse Document Frequency (TF-IDF)

### 2.4.1 Introduction

En faisant nos recherches nous nous sommes rendu compte de certaines limites imposées par la méthode du PageRank. Il est notamment impossible d'afficher les résultats de la recherche en fonction des mots renseignés par l'utilisateur dans la barre de recherche. Le score du PageRank se contente de donner plus de poids aux sites ayant beaucoup de liens entrants et peu de liens sortants.

Il a donc été nécessaire de créer une méthode calculant un score variant en fonction des mots clés de la recherche. Cette méthode s'appelle "Term frequency and inververse document frequency" (TF-IDF).

### 2.4.2 Principe

Le principe de base de cette méthode est assez simple. Nous calculons la fréquence d'apparition d'un mot rentré dans la barre de recherche dans chaque page et nous le multiplions par le logarithme du nombre de sites total divisé par le nombre de sites dans lesquels ce mot apparaît pour chaque mot clé. Nous avons donc la formule suivante :

$$\frac{n}{\sum_{mot} n_{mot}} * \log \frac{|P|}{|\{p, mot \in p\}|}$$

avec P l'ensemble des pages et p une page

### 2.4.3 Application

Pour appliquer le principe décrit précédemment, il nous suffit de modifier légèrement le crawler. En plus de récupérer tous les liens sur la page, il doit également extraire tous les mots et les stocker dans un dictionnaire. Un dictionnaire est une structure de données très utilisée en Python, permettant de stocker divers types d'informations. Dans notre cas, nous utiliserons le lien du site analysé comme clé principale et stockerons en valeur un sous-dictionnaire contenant chaque mot et son nombre d'apparitions.

Cependant, cette solution n'est pas la plus optimale, car elle nous obligera à parcourir toutes les pages lors du calcul du score. Pour simplifier cela, nous devons transformer ce dictionnaire en un système de référencement inverse. Dans ce nouveau dictionnaire, chaque mot sera la clé principale, et pour chaque mot, nous stockerons les sites où il apparaît ainsi

que son nombre d'apparitions. Cette transformation simplifie grandement la complexité de nos calculs.

## 2.5 Travaux en vue de la présentation

### 2.5.1 Introduction

Pour pouvoir aborder le calcul du PageRank, il est important de comprendre certaines notions essentielles. Cet algorithme essaye de prévoir le comportement de l'humain. Alors, dans un but de vulgariser le projet pour l'oral et d'avoir une représentation visuelle, nous avons entrepris de réaliser plusieurs supports qui illustrent le comportement humain.

### 2.5.2 Animation

Le concept de surfeur aléatoire est une notion cruciale du PageRank, cependant, c'est une idée peu compréhensible au premier abord. C'est pour cela que Paul K. a décidé de réaliser une animation interactive pour la présentation de ce concept. Il a pour cela utilisé le moteur de jeu Unity et le langage de programmation C#. Cette première simulation permet d'illustrer le parcours du graphe par le surfeur aléatoire, d'abord sur un graphe "normal", puis sur un deuxième avec des Spider Traps, afin d'introduire le facteur d'amortissement. Les paramètres de la simulation tels que la vitesse ou le facteur d'amortissement peuvent être modifiés via l'interface graphique.

Une deuxième simulation permet également de visualiser le comportement d'un crawler, qui explore un graphe simple généré aléatoirement, comportant environ 40 sommets.

### 2.5.3 Sites web

Pour mieux comprendre le calcul du PageRank, il est également important de l'illustrer dans un contexte plus concret. Pour cela, Gauvain a développé 6 sites web différents sur le thème des tomates. Ces pages possèdent des hyperliens redirigeant vers un ou plusieurs des 6 sites. Un hyperlien dans une page peut être défini comme un texte ou un bouton sur lequel on peut appuyer pour se rediriger vers une autre page web. Le langage HTML a été utilisé pour construire les sites web, il s'agit d'un langage de description, du code CSS a également permis de définir les éléments visuels tels que les couleurs et les formes par exemple.

Pour suivre les interactions de l'utilisateur avec les sites, du code dans le langage de programmation JavaScript a également été implémenté pour afficher en temps réel, dans le coin de chaque page, l'évolution de la probabilité qu'une page soit visitée en fonction des choix de l'utilisateur. Ce point a posé de nombreux problèmes car tout d'abord un seul programme doit interagir avec les 6 sites web et de plus, les données doivent être stockées de manière à ce que les compteurs ne soient pas réinitialisés à chaque page web visitée.

Ces sites web apparaissent sur le moteur de recherche évoqué dans la prochaine partie. L'utilisateur a la possibilité de revenir sur la page du moteur de recherche, ce qui décrit le comportement du surfeur aléatoire.

### 2.5.4 Moteur de recherche

En effet, Nous avons mis en place un moteur de recherche similaire à Google. Cela consiste en une barre de recherche sur laquelle nous pouvons renseigner des mots clés ou

une phrase. Lorsque nous appuyons sur rechercher, les sites apparaissent les uns à la suite des autres en fonction de leur score total.

Pour le fonctionnement, lorsque nous appuyons sur rechercher, Nous récupérons la suite de mots pour calculer le TF-IDF. C'est en utilisant le score Page Rank ainsi que le score TF-IDF que nous pouvons calculer le score total qui est simplement le calcul suivant :

$$ScoreFinal = 0.7 * TF - IDF + 0.3 * PageRank$$

# Conclusion et perspectives

- Le PageRank a été une idée novatrice qui a permis de propulser Google en tant que numéro 1 des moteurs de recherche puisqu'il a réussi à utiliser des notions qui n'étaient pas exploitées pour la pertinence des recherches, ces notions étant notamment les hyperliens, qui sont récupérés grâce au crawler. Le calcul du PageRank s'est aussi optimisé au fil du temps, en prenant en compte d'autres paramètres, tels que le temps passé sur la page, ou encore l'amélioration du surfeur aléatoire : le surfeur raisonnable. Ce dernier fait des différences entre les liens. Maintenant, la position du lien sur la page a de l'importance, ou encore ce que va référencer le lien. Tout cela augmente ou réduit la probabilité qu'un crawler le prenne en considération.
- Nous pouvons affirmer que lors de ce projet encadré nous avons acquis de nouvelles compétences en programmation puisque l'étude de cet algorithme nous a amené à étudier de nouveaux langages informatiques. Durant ce projet, l'ensemble du groupe a du faire preuve d'autonomie et d'esprit critique face aux recherches que nous avons menées. Le projet nous a aussi permis d'apprendre à mieux nous organiser et à répartir le travail au sein d'un groupe conséquent. La communication fut un élément clé de ce travail.
- Pour la poursuite de ce projet, nous pourrions, comme dit précédemment, étudier différentes améliorations possibles de l'algorithme : le surfeur raisonnable, l'implémentation de mesures pour empêcher les sites de gonfler leur score... Nous pourrions également chercher des manières d'optimiser le calcul du PageRank, d'améliorer la vitesse des crawlers, ...

# Bibliographie

[1] STANFORD INFOLAB http://infolab.stanford.edu/~backrub/google.html (Valide à la date du 13/06/2024)

[2] CRÉATEUR 2 SITE https://createur2site.fr/seo/fonctionnement-google/algorithmes/pagerank/ (Valide à la date du 13/06/2024)

[3] WIKIPÉDIA - WORLD WIDE WEB WANDERER https://en.wikipedia.org/wiki/World_Wide_Web_Wanderer (Valide à la date du 13/06/2024)

[4] WIKIPÉDIA - WEB CRAWLER https://en.wikipedia.org/wiki/Web_crawler (Valide à la date du 13/06/2024)

[5] MIT http://www.mit.edu/%7Emkgray/net/ (Valide à la date du 13/06/2024)

[6] WIKIPÉDIA - WORLD WIDE WEB https://fr.wikipedia.org/wiki/World_Wide_Web (Valide à la date du 13/06/2024)

[7] WIKIPÉDIA - TF-IDF https://fr.wikipedia.org/wiki/TF-IDF (Valide à la date du 13/06/2024)

[8] Paolo Boldi, Massimo Santini, Sebastiano Vigna
PAGERANK AS A FUNCTION OF THE DAMPING FACTOR https://vigna.di.unimi.it/ftp/papers/PageRankAsFunction.pdf (Valide à la date du 13/06/2024)

[9] TOWARDS DATA SCIENCE https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af (Valide à la date du 13/06/2024)

# Table des figures

# Annexe A

# Partie théorique et mathématiques

## A.1 Surfeur aléatoire

Le score PageRank est déterminé en passant par un modèle : le surfeur aléatoire. Il correspond à un utilisateur qui se déplacerait sur le web en cliquant aléatoirement sur les liens de la page où il se trouve. Nous pouvons modéliser les liens entre les pages par un graphe orienté. Le surfeur aléatoire va parcourir ce graphe, et nous allons chercher à déterminer la probabilité que ce surfeur se trouve sur une page à l'étape $i$, connaissant cette probabilité pour l'étape $i + 1$.

Pour calculer la probabilité à l'étape $i$, chaque page va propager équitablement son score entre les pages vers lesquelles elle pointe (grâce à des liens). On peut représenter les liens du graphe par une matrice carrée de taille $N$, avec $N$ le nombre de sites considérés, où le $x$-ème coefficient de la $y$-ème ligne représente la probabilité que ce surfeur aléatoire passe du site numéro $y$ au site numéro $x$. Cette probabilité vaudra donc $0$ si $y$ n'a pas de lien vers $x$, et $\frac{1}{n}$ sinon, où $n$ est le nombre de liens sur le site $y$ (puisque nous avons une chance égale de cliquer sur chaque lien). Nous appellerons cette matrice $P$.

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

Ci-dessus, un exemple de matrice de transition $P$ de taille $6 * 6$, nous considérons donc 6 sites. Le graphe associé à cette matrice est affiché en 2.1.4.

Pour représenter le PageRank des sites à chaque étape, nous aurons une matrice ligne de longueur $N$, que nous appellerons $V$. Les PageRank à chaque étape correspondront aux termes d'une suite de matrices définies par récurrence, avec $V_{i+1} = V_i P$.

Cette suite est convergente, et se rapprochera donc à chaque étape de sa limite, et c'est cette limite que nous retiendrons comme le PageRank final.

## A.2 Spider Traps et facteur d'amortissement

# Annexe B

# Partie théorique et mathématiques

## B.1 Introduction et Chaînes de Markov

Commençons par poser E un espace dénombrable. Tous les éléments $x \in E$ sont des états. On appelle mesure un vecteur ligne $V = (V_x, x \in E)$ où tous les $V_x$ sont des réels positifs ou nuls. Si la somme des $V_x$ d'une mesure vaut 1, alors cette mesure est une probabilité ou distribution.

La matrice de transition est une matrice carrée d'une application de $E^2$ dans $[0, 1]$ où la somme des coefficients d'une ligne est égale à 1. On dit que cette matrice est stochastique.

Une chaîne de Markov est une suite de variables aléatoires $(X_n)_{n\in\mathbf{N}}$ à valeurs dans E où, pour tout $n \in \mathbf{N}$, la loi conditionnelle de $X_{n+1}$ sachant $X_0, ..., X_n$ est égale à sa loi conditionnelle sachant $X_n$. Cela se traduit par :

$\mathbf{P}(X_{n+1} = y_{n+1}|X_0 = y_0, ..., X_n = y_n) = \mathbf{P}(X_{n+1} = y_{n+1}|X_n = y_n)$

Autrement dit, une chaîne de Markov est une chaîne d'états où, pour déterminer l'état suivant, seule la connaissance de l'état actuel est nécessaire. Les chaînes de Markov sont "sans mémoire". Pour connaître la probabilité de passer d'un état à un autre, il suffit de lire la matrice de transition $P$. On a :

$\mathbf{P}(X_{n+1} = y|X_n = x) = P_n(x, y)$

On dit qu'une mesure est invariante si c'est un point fixe de l'application définie par E, autrement dit $V$ est invariante si $V = VP$, où $P$ est notre matrice de transition. On peut interpréter une mesure comme étant la mesure des probabilités de se trouver dans chaque état de E à une certaine étape. Pour obtenir la mesure de l'étape suivante, il suffit alors juste de multiplier la mesure par la matrice de transition : $V_{n+1} = V_nP$

Le comportement du surfeur aléatoire peut être modélisé par une chaîne de Markov. En effet, le déplacement du surfeur d'un site vers un autre ne dépend que du site sur lequel il est, ce qui correspond à la propriété principale d'une chaîne de Markov. On considérera chaque site comme un état, et donc $E$ est de dimension finie égale à $N$, le nombre de sites sur notre graphe.

Ces définitions posées, nous allons pouvoir étudier quelques unes des propriétés de notre chaîne de Markov.

## B.2 Irréductibilité

On dit d'une chaîne de Markov qu'elle est irréductible si, pour tout couple $(x, y) \in E^2$, il existe un entier $k = k(x, y)$ et une suite finie $x = x_0, x_1, ..., x_{k-1}, x_k = y$ tels que $P(x_i, x_{i+1}) > 0$ pour $i = 0, ..., k - 1$

Cela peut se traduire par : $\mathbf{P}(X_k = y|X_0 = x) = P^k(x, y) > 0$
Autrement dit : en partant de n'importe quel état $x$, il existe au moins un chemin, de longueur $k$ exactement, qui permet d'aller à n'importe quel autre état $y$.

Trivialement, la chaîne de Markov de notre algorithme PageRank est irréductible. En effet, rappelons qu'il existe notre facteur d'amortissement $\beta \in [0, 1[$, et la probabilité de se téléporter d'un état (site) à n'importe quel autre est de $\frac{1-\beta}{N}$, d'où pour tout $(x, y) \in E^2$, on a pour $k = 1$, $\mathbf{P}(X_1 = y|X_0 = x) = P(x, y) \geq \frac{1-\beta}{N} > 0$. Notre chaîne de Markov est donc irréductible.

Un théorème nous permet également d'affirmer que, pour $E$ un espace fini de cardinal $d$, l'ensemble des mesures invariantes pour la matrice de transition $P$ de taille $d * d$, est un sous-ensemble non vide, compact et convexe de l'ensemble des mesures sur $E$.
Ce qui nous intéresse ici est le fait que l'ensemble des mesures invariantes est non vide. Puisque dans le cas de PageRank, $E$ est de dimension finie $N$, nous avons donc l'existence d'au moins une mesure invariante. Cependant, nous n'avons pas encore montré que cette mesure est unique.
L'unicité nous viendra d'un autre théorème, qui affirme que si la chaîne de Markov est irréductible, $P$ admet au plus une probabilité invariante.
Nous avons justifié ici que notre chaîne de Markov est irréductible, la probabilité invariante est donc unique.
Nous avons donc bien l'existence et l'unicité de cette probabilité invariante.

## B.3 Périodicité

Soit $x \in E$. On pose $R(x) = \{n \in \mathbf{N}^*, P^n(x, x) > 0\}$ l'ensemble des temps de retour en x. Un temps de retour correspond à la longueur d'un chemin permettant, en partant de x, d'y revenir. La période $p(x)$ de $x$ est le PGCD de $R(x)$.
Nous savons également que tous les états de $E$ ont la même période si la chaîne est irréductible. Cette période commune est celle de la chaîne. On dira que la chaîne est apériodique si cette période est de 1.

Dans notre cas, nous savons que la chaîne est irréductible. De plus, par un même argument utilisé plus tôt, nous avons $\mathbf{P}(X_1 = y|X_0 = x) = P(x, y) \geq \frac{1-\beta}{N} > 0$ pour tout $(x, y) \in E^2$, et en particulier si $x = y$. Ainsi, nous avons $1 \in R(x)$, donc nécessairement son PGCD vaut 1 et la chaîne est apériodique.

## B.4 Récurrence

Une chaîne de Markov irréductible sur un espace fini $E$ est récurrente. De plus, une chaîne de Markov irréductible et récurrente qui possède une unique distribution invariante est dite récurrente positive.
La chaîne de Markov de l'algorithme PageRank vérifie toutes ces propriétés, elle est donc bien récurrente positive.

## B.5 Convergence

Une chaîne de Markov, dans un espace $E$ fini, si elle est irréductible, récurrente positive et apériodique, verra la suite $P^n$ (avec $P$ la matrice de transition) converger vers la matrice

$\pi$. Cette matrice $\pi$ est la matrice dont toutes les lignes sont égales à l'unique probabilité invariante.

Notre chaîne de Markov vérifie bien ces propriétés, elle est donc convergente. En particulier, l'unique probabilité invariante vers laquelle vont converger les mesures de la chaîne de Markov, est indépendante de la distribution initiale. (On rappelle qu'une distribution est un vecteur ligne de longueur $N$ dont la somme des coefficients vaut 1)

## B.6 Conséquences de l'absence du facteur d'amortissement

Comme vu plus tôt, le facteur d'amortissement permet de rendre triviale la justification de la convergence de notre chaîne de Markov. Mais quelles conséquences seraient causées par son absence ?

Commençons par poser la notion d'accessibilité. Un état $y \in E$ est accessible depuis $x \in E$ s'il existe $n \in \mathbf{N}$ tel que $\mathbf{P}_x(X_n = y) > 0$. Autrement dit, il faut qu'il y ait au moins un chemin de longueur $n \geq 0$ qui permet de passer de $x$ à $y$. On note $x \to y$.

Deux états x et y communiquent si $x \to y$ et $y \to x$. On note alors $x \leftrightarrow y$. Cette relation de communication est transitive, c'est à dire que si $x \leftrightarrow y$ et $y \leftrightarrow z$, alors $x \leftrightarrow z$. La notion de communication permet de définir des classes d'équivalence, qui sont des partitions de E.

On dira qu'une classe C est close si $x \in C$ et $x \to y \Rightarrow y \in C$. Concrètement, une classe close est une classe dont "on ne peut pas sortir". Les Spider Traps, et les impasses, sont des classes closes.
En particulier, si notre espace $E$ des sites comporte une classe close $C \neq E$ , alors cela signifie qu'il existe $(x, y) \in E^2$, avec $x \in C$, tel que $y$ n'est pas accessible depuis $x$, et donc notre graphe n'est pas irréductible. Or, rappelons que c'est justement la propriété d'irréductibilité qui permet d'affirmer l'unicité de la distribution stationnaire, mais l'existence des distributions stationnaires n'est pas remise en cause (puisqu'elle ne dépend que du caractère fini de E).

On peut affirmer qu'il y a autant de distributions stationnaires qu'il y a de classes closes dans la chaîne de Markov. En effet, la restriction de la chaîne de Markov à une classe close $C$ est une chaîne de Markov d'espace d'états $C$. Et si cette restriction est irréductible, elle admet elle-même une unique distribution stationnaire.
Il nous est donc possible de diviser l'espace d'états fini $E$ en un nombre fini et non nul de classes closes $C$, admettant chacune une unique distribution stationnaire (qui est évidemment différente des distributions stationnaires des autres classes, ces classes closes étant disjointes). Il reste donc également un certain nombre, potentiellement nul, de classes transientes (c'est à dire non closes), et ces classes n'admettent pas de distributions stationnaires, car elles ne possèdent que des états transitoires.

La distribution stationnaire vers laquelle nous convergerons dépend alors de la distribution initiale (il est par exemple évident que, partant d'un état situé dans une classe close, nous convergerons évidemment vers la distribution stationnaire associée à celle-ci).

# Annexe C

# Annexe Code

## C.1 Exemples de code Python

**Programme 1 : Fonction Seek**

Fonction qui lorsqu'on lui donne un URL récupère l'ensemble des liens présents dans la page et les ajoute à une liste

```
def seek(url: str) -> list:
    try:
        urls = []
        current_url = url
        #envoi de la requ te a l'url
        response = requests.get(current_url)
        #creation du parser
        soup = BeautifulSoup(response.content, "html.parser")
        #recuperation des hyperliens
        link_elements = soup.select("a[href]")[:10]
        for link_element in link_elements:
            url = link_element['href']
            #verification de la validite des liens et ajout dans la liste
            if url and not url.startswith('#') and (url.startswith('http')
                if url[0] == '/':
                    url = current_url[:-1] +  url
                urls.append(url)
        return np.unique(urls)
    except requests.exceptions.RequestException as e:
        print('Error:', e)
        return []
```

**Programme 2 : Fonction Ajout**

Fonction qui à partir d'un URL va exécuter la fonction Seek et ajouter la liste de liens obtenus dans le dictionnaire stockant toutes nos informations : la toile

```
def ajout_toile(url_original: str, toile: dict, queue: list):
    #ajoute l'url dans le dictionnaire avec ses liens sortants
    toile[url_original] = seek(url_original)
    for lien in toile[url_original]:
        if lien not in queue:
```

```
            #ajoute les liens qui n'y sont pas dans la queue
            queue.append(lien)
    return toile, queue
```

**Programme 3 : Calcul Score TF-IDF**

Fonction qui à partir de la toile générée par ajout du nombre d'apparitions de chaque mot dans chaque page et du nombre de pages, calcule le score TF-IDF en fonction des mots dans la barre de recherche

```
def calculation_score(toile, score_total, TF_score, IDF_score,
    sites_count_word_count1, mot_recherche, nb_page, constante_tp):
    # IDF
    for mot in mot_recherche:
        if mot is not None:
            nb_site_pour_mot = nombre_de_sites(
                sites_count_word_count1, mot)
            if nb_site_pour_mot > 0:
                IDF_score[mot] = np.log10(nb_page / nb_site_pour_mot)
            else:
                IDF_score[mot] = 0
            #print(f"idf_score de : {mot} = {IDF_score[mot]}")
    # TF + total
    for site, liens in toile.items():  # Parcourir le dictionnaire toile
        TF_IDF_global = 0
        # Nombre total de mots dans la page
        nb_mots_dans_page = count_word(site)
        for mot in mot_recherche:
            if mot is not None:
                if mot in sites_count_word_count1:
                    # Calcul du TF pour ce mot et ce site
                    nb_occurence = sites_count_word_count1[mot][site] if
                    site in sites_count_word_count1[mot] else 0
                    TF_score[mot] = nb_occurence / nb_mots_dans_page if
                    nb_mots_dans_page > 0 else 0
                    #print(f"tf_score de : {mot} pour le site {site} =
                    # {TF_score[mot]}")
                else:
                    TF_score[mot] = 0
                # Ajouter le score TF-IDF pour ce mot au score
                # global du site
                TF_IDF_global += TF_score[mot] * IDF_score[mot]
        # Calculer le score PageRank pour le site
        matrice_PG = matrice_page_rank(toile, nb_page, constante_tp)
        # Calculer le score total pour le site
        # Utiliser l'indice de la cl  dans la liste des cl s
        score_total[site] = 0.7 * TF_IDF_global +
        0.3 * matrice_PG[list(toile.keys()).index(site)]
    return score_total
```

**Programme 4 : Implémentation du produit matriciel et récupération du crawler**

Programme qui à partir du crawler, récupère les sites de la base de données et les organisent dans un dictionnaire. Depuis ce dictionnaire, représentant la toile, sont créées nos différentes matrices d'initialisation puis effectue le produit matriciel jusqu'à un certain seuil de convergence.

```
import numpy as np

from crawler import fill_toile
graph = {}
graph = fill_toile(graph)
print(graph)
constante_tp = 0.85
nbsites = len(graph)
x=y=nbsites

def lire_lien (graph : "toile", cle_reference : "site") -> int :
    nombre_site_ref = 0
    colonnes = []
    for site in graph[cle_reference]:
        compteur = 0
        url = site
        for key in graph:
            if url == key:
                nombre_site_ref += 1
                colonnes.append(compteur)
            compteur += 1

    return nombre_site_ref, colonnes

def init_matrice_ligne(graph : "toile") -> "array":
    matrice_ligne = np.full(nbsites, 1/nbsites)
    return matrice_ligne

def init_grosse_matrice(graph : "toile") -> "array":
    surfeur = np.full((nbsites, nbsites), (1-constante_tp)/nbsites)
            #forme de la matrice     #valeur remplie pour chaque element
#[[(1-constante_tp)/nbsites for i in range (x)] for j in range (y)]
#on vas pas utiliser  a  car numpy plus rapide #[[Valeur for i in range(9)]
#vas cree 9 fois le meme objet c'est nul ici
    ligne = 0
    for cle_reference in graph:
        nb_sites_ref,colonnes = lire_lien(graph,cle_reference)
        for c in colonnes:
            surfeur[ligne][c] += constante_tp/nb_sites_ref
        ligne += 1
    return surfeur

petite_matrice = init_matrice_ligne(graph)
resultat = petite_matrice
```

```
print(f"Matrice au depart initialise :  {resultat} \n")
nbiteration = 0
surfeur = init_grosse_matrice(graph)
diff = np.inf
while diff > 0.0001:
    matrice_precedente = resultat.copy()
    nbiteration +=1
    print(f"Apr s {nbiteration} it ration on a :  {resultat}")
    resultat = resultat @ surfeur
    diff = np.max(np.abs(resultat - matrice_precedente))
print("Matrice Final :")
print(resultat)
```