Moteurs de recherche (Master 2)

TP 1 : Programmer un *PageRank*

Cours : Michel Habib / TP : Sylvain Perifel

Les TP se font **en binôme** en C, en C++ ou en Java, à l'exclusion de tout autre langage.
La note de TP compte pour la moitié de la note finale. Ce sont vos rendus qui sont évalués.
Vos travaux doivent être soumis sur Didel (`http://didel.script.univ-paris-diderot.fr`)
le **11 février** au plus tard.
S'inscrire au cours "Moteurs de recherche" sur Didel.
Soumettez une **archive** `Nom1-Nom2.tgz` ou `Nom1-Nom2.zip` de vos sources.
N'oubliez pas de rendre aussi un rapport contenant vos réponses aux questions du TP !

1 Un format simple pour les graphes et les matrices creuses

Soit n un entier. On veut stocker des matrices réelles $n \times n$, avec ces deux contraintes :
- accès facile à la ligne i de la matrice ;
- on ne stocke pas les zéros.

Soit M une matrice ayant m entrées non nulles. On code M sous forme de :
- l'entier n ;
- un tableau C de m `float` (NB : un stockage simple précision est suffisant pour nos calculs) ;
- un tableau I de m `int` (suffisant pour nos calculs) ;
- un tableau L de $(n+1)$ `int`,

qui vérifient les règles suivantes :
- la première ligne (respectivement colonne) est la ligne (resp. colonne) numéro 0 ;
- C contient les contenus de toutes les cases non nulles de la matrice M : d'abord ceux de la première ligne de M, puis ceux de la deuxième ligne, etc. ;
- $L[i]$ est l'indice du début de la i-ème ligne de M dans C : cette ligne s'étend donc de $C[L[i]]$ inclus à $C[L[i+1]]$ exclu ;
- mais les éléments non nuls de la ligne i de la matrice M ne sont pas nécessairement consécutifs : afin de pouvoir retrouver leur place et que le codage avec ces trois tableaux soit un codage exact de la matrice, on utilise le tableau I, qui contient les indices des colonnes associées aux éléments non nuls de la ligne i ;
- si $C[k]$ est la j-ème case de la ligne i alors $I[k] = j$;
- $L[n+1] = m$ (afin de traiter la dernière ligne comme les autres) ;
- si la ligne i ne contient que des zéros on a $L[i] = L[i+1]$.

Exemple de matrice M :

0	3	5	8
1	0	2	0
0	0	0	0
0	3	0	0

tableau L :

0	3	5	5	6

tableau C :

3	5	8	1	2	3

tableau I :

1	2	3	0	2	1

Exercice 1

Définir les types matrice et graphe, et coder une instance de matrice (par exemple celle ci-dessus) dans le format ci-dessus. Programmer le produit d'une matrice M par un vecteur V, en utilisant le codage défini ci-dessus, et vérifier le résultat.
Évaluer la complexité du calcul.

2 Pagerank-zero

Soit G un graphe orienté fortement connexe et apériodique (le pgcd de la taille de ses circuits vaut 1). La matrice stochastique équiprobable associée à G est une matrice M telle que

— $M_{ij} = 0$ si (i, j) n'est pas un arc de G ;
— et si (i, j) est un arc alors $M_{ij} = 1/d^+(i)$ (où $d^+(i)$ désigne le degré sortant du sommet i).

Exercice 2

Nous voulons faire le calcul non pas de $P = MV$ (cf. exercice 1) mais de $P = M^tV$. Or transposer une matrice est coûteux. Nous allons programmer M^tV **sans calculer explicitement** la transposée de M :

$$P[i] = \sum_{j=0}^{j=n-1} M_{ji}V[j].$$

Calculer $P[i]$ revient à parcourir une colonne de M. Mais si on parcourt la **ligne** i on peut faire $P[j]$ += $M_{ij}V[i]$ en ayant initialisé P au vecteur nul. On fait m fois de suite ces incréments.

Programmer cela, en parcourant donc la matrice M ligne par ligne. Le calcul doit **impérativement** se faire en $O(n + m)$ et en une seule passe des tableaux L, C et I.

Exercice 3

Rappelons l'algorithme standard de PageRank.

Données : une matrice M, une distribution de probabilités Z et un réel $\epsilon > 0$.
Résultat : le vecteur P de pagerank-zero avec une précision ϵ.
début
  $P_0 = Z$;
  répéter
    $P_{k+1} = M^t P_k$;
    $\delta = \|P_{k+1} - P_k\|_1$;
  jusqu'à $\delta < \epsilon$;
fin

En vous servant de l'exercice précédent, écrire un programme qui demande un sommet de départ et un nombre de pas et affiche, étape par étape, la probabilité d'être sur les différents sommets. Vérifier pour le graphe donné en exemple que vous obtenez bien les mêmes vecteurs.

En déduire un algorithme qui calcule le PageRank-zero, en partant du sommet 0 (on a donc $Z = (1, 0, ...0)$) et en faisant k pas de calcul.

Exercice 4

1. Testez votre algorithme sur un ensemble de graphes de votre choix.

   Analyser les résultats : convergence ou non, rapidité de la convergence...

   Piste : vous pouvez commencer par appliquer une centaine d'itérations avec des vecteurs initiaux différents sur vos graphes puis expliquer les résultats obtenus en fonction des résultats du cours.
2. Le paramètre Z est une distribution de probabilités, c'est-à-dire un vecteur à coefficients dans $[0, 1]$ et dont la somme des coefficients vaut 1. Que représente ce vecteur ? Expérimentez plusieurs valeurs. Qu'observez-vous ?
3. Expérimentez cet algorithme sur une clique, puis sur un graphe symétrique. Que remarquez-vous dans ce dernier cas ?

3 Pagerank avec facteur zap

Exercice 5

Dans le modèle du surfeur, celui-ci peut choisir : soit de suivre un lien sortant avec un probabilité $(1 - d)$, soit « zapper » sur une page aléatoire avec probabilité d. On propose de prendre le facteur zap d compris entre 0,1 et 0,2.
On programmera la variante incluant le zap (cf. cours) :

$$P_{k+1}[i] = \frac{d}{n} + (1 - d) \sum_{j=0}^{n-1} M_{j,i} P_k(j).$$

1. Tester ce nouvel algorithme sur le même ensemble de graphes qu'à l'exercice précédent. Comparer les résultats, et essayer plusieurs valeurs de zap.
2. Essayer maintenant vos algorithmes sur des plus grandes données : on pourra utiliser les grands graphes de `http://snap.stanford.edu/data/index.html`.

4 Conseils pour le rendu

Le rapport est un élément **primordial** du rendu. Un document au format `pdf` est attendu (longueur indicative : 5 à 10 pages). Il doit être rédigé avec soin et comporter les réponses aux questions et des discussions sur les choix réalisés et sur les résultats obtenus. Ne pas oublier d'évaluer la complexité de vos algorithmes.

Vos codes source doivent être commentés. Les algorithmes et les choix de programmation doivent être expliqués brièvement dans le rapport.

Enfin, on vous demande **d'expérimenter** vos algorithmes sur des graphes variés et d'analyser les résultats, le temps de calcul, etc.