TP2 - Branching pour le stable maximum

Optimisation et Recherche Operationnelle

M1 Info - semestre d'automne 2020-2021

Université Claude Bernard Lyon 1

Christophe Crespelle
christophe.crespelle@inria.fr

Le but de ce TP est d'implementer l'algorithme de branching vu en cours pour la recherche d'un stable maximum dans un graphe. Deux fichiers sont fournis : `main.c` et `auxiliaire.c`, le second etant incorpore par un `#include` au debut de `main.c`. Le fichier `main.c` contient une trame de l'architecture generale du programme et le fichier `auxiliaire.c` contient des fonctions que vous pourrez utiliser, pour manipuler des graphes et des listes de noeuds. Ce dernier est le meme que celui fourni pour le TP1 a la difference qu'il contient deja les procedures `free_nodl` et `free_graph`. Deux autres nouvelles fonctions vous sont egalement fournies : `vider`, qui vide une liste (sans la liberer), et `copy_append`, qui copie le contenu d'une liste et le place a la fin d'une autre. Vous coderez les fonctions necessaires a l'algorithme dans un troisieme fichier que vous creerez et que vous incluerez de la meme maniere au debut de `main.c`.
Le programme se compile grace a la commande

```
gcc -Wall -Wextra -O3 main.c -o stabmax
```

et s'execute en tapant `./stabmax`. Tapez `./stabmax -h` pour afficher l'aide. Vous pourrez tester votre algorithme sur les memes fichiers de donnees qu'au TP1. En particulier, pour la phase de conception de l'algorithme, verifiez vos resultats sur les deux graphes jouets.

1 Implementation de l'algorithme

Vous implemeterez toutes les fonctions necessaires pour l'algorithme dans un fichier a part, nomme par exemple `fonctions-algo.c`, que vous appelerez au debut de `main.c` par un `#include`.

1.1 Graphe induit

L'algorithme vu en cours repose sur une fonction recursive, que nous appelerons `calcule_stable_max`, qui a chaque appel recursif retire du graphe :

- un sommet x, qu'elle place dans le stable courant S, et
- les voisins de x, car ils ne peuvent pas participer a un stable contenant x.

Ainsi, chaque appel a la fonction `calcule_stable_max` opere sur un sous-graphe induit du graphe de depart G.
Afin de simplifier l'implementation de l'algorithme, nous n'allons pas explicitement former le graphe induit courant. C'est a dire que la structure representant le graphe G donne en entree ne sera jamais modifiee au cours de l'algorithme. Pour representer le graphe induit courant sur lequel nous travaillons a une etape de l'algorithme, nous allons en plus du graphe G utiliser deux tableaux. Le premier que vous pourrez nommer `presence` sera indice par les identifiants des sommets du graphe (de 0 a $n-1$) et contiendra un booleen :

- vrai pour indiquer que le sommet est present dans le graphe induit courant,
- faux pour indiquer qu'il a deja ete retire du graphe (lors des appels recursifs precedents).

Le deuxieme tableau, nomme `degre_courant`, sera aussi indice par les identifiants des sommets du graphe (de 0 a $n-1$) et contiendra des entiers. Il indiquera, pour chaque sommet y present dans le graphe induit courant $\tilde{G}$, quel est le degre de y dans $\tilde{G}$. Pour les sommets z qui ne sont pas presents dans $\tilde{G}$, la valeur de `degre_courant[z]` pourra etre quelconque, a vous de la choisir comme cela vous convient le mieux.
Remarquez que le tableau `degre_courant` n'est pas necessaire pour representer le graphe $\tilde{G}$, le tableau `presence` suffit dans ce but. Le tableau `degre_courant` permet en revanche de selectionner un sommet y de degre minimum dans $\tilde{G}$ sans avoir a parcourir toutes les listes d'adjacences du graphe G. En effet, souvenez vous qu'a chaque etape, l'algorithme choisit le nouveau sommet x a placer dans le stable parmi y et les voisins de y dans $\tilde{G}$, pour un sommet y de degre minimum dans $\tilde{G}$.

Question 1.
Declarez les tableaux `presence` et `degre_courant` dans la section `DATA STUCTURE` du fichier `main.c`. Reservez l'espace necessaire sur le tas pour ces deux tableaux, a l'aide de la fonction `malloc()` (reprenez l'annexe qui lui est dediee dans le TP1 si besoin), et initialisez leur contenu comme il doit l'etre au debut de l'algorithme. Au debut, on a $\tilde{G} = G$, donc tous les sommets sont presents et leur degre courant est exactement leur degre dans G.

Question 2. Dans le fichier `fonctions-algo.c`, ecrivez une fonction qui prend trois parametres, un tableau de presence, un tableau de degres courants et la taille de ces deux tableaux, et qui retourne un sommet de degre courant minimum parmi les sommets presents.

1.2 Branching

La profondeur maximum d'une branche de recursivite de la fonction `calcule_stable_max` peut etre de l'ordre du nombre de sommets n dans le graphe de depart. Dans ces conditions, il est inenvisageable de stocker une copie des tableaux `presence` et `degre_courant` pour chaque appel a la fonction `calcule_stable_max`. Cela prendrait trop de place en memoire et limiterait de fait la taille des graphes que l'on pourrait traiter.
Afin d'eviter cet ecueil, tous les appels recursifs a la fonction `calcule_stable_max` devront travailler sur les memes tableaux `presence` et `degre_courant`, qui seront des variables globales de votre programme. Chaque appel a `calcule_stable_max` se verra passer en parametre un pointeur vers chacun de ces deux tableaux. Il est donc primordial que

vous preniez soin, **a la fin de chaque appel, de remettre les deux tableaux, ainsi que les autres variables globales pour lesquelles cela est necessaire, dans l'etat exact dans lequel vous les avez trouves au debut de l'appel**.
Vous procederez de meme pour la liste des sommets dans le stable courant : elle sera une variable globale du programme, vous passerez un pointeur vers cette variable a `calcule_stable_max` et vous la remettrez en etat a la fin de chaque appel a `calcule_stable_max`.
Enfin, n'oubliez pas qu'au cours de l'algorithme, vous devrez stocker et mettre a jour :

- la taille du plus grand stable rencontre jusqu'a present et
- la liste des sommets dans ce stable.

La encore, vous ferez le choix de stocker ces informations dans des variables globales et vous donnerez a `calcule_stable_max` un pointeur vers ces variables comme parametre.

Question 3. Implementez la fonction `calcule_stable_max` en lui passant explicitement en parametre des pointeurs vers toutes les variables globales du programme qu'elle utilise.

Indication. La fonction `calcule_stable_max` aura parmi ses parametres le sommet x que l'on a decide de mettre dans le stable courant. Le travail effectue par cette fonction suivra les grandes etapes ci-dessous :

- retirer x et ses voisins du graphe courant
- selectionner un sommet y de degre courant minimum dans le nouveau graphe ainsi obtenu
- faire les appels recursifs a `calcule_stable_max` pour chaque sommet parmi y et ses voisins
- remettre x et les voisins de x dans le graphe courant (en vue des prochains appels a `calcule_stable_max`)

Il pourra etre utile de stocker la liste des voisins que x avait dans le graphe courant juste avant qu'on ne retire x et ses voisins.

1.3 Condition d'arret

Pour l'instant, le cas terminal de la fonction recursive `calcule_stable_max` que vous avez ecrite est le cas ou le graphe induit courant ne contient aucun sommet. C'est a ce moment que vous devez verifier si le stable courant est de taille superieure au stable maximum trouve jusqu'a lors et que vous devez actualiser ce dernier si c'est le cas. Afin d'ecourter certaines branches de l'arbre des appels recursifs, on peut s'arreter sous une condition plus faible (c'est a dire plus souvent satisfaite).

Question 4. Sous quelle condition simple impliquant le nombre de sommets dans le stable courant et le nombre de sommets dans le graphe induit courant peut on etre sur que la branche en cours ne permettra pas de depasser la taille du stable maximum trouve jusqu'a lors ?

Vous pourrez prendre des variables globales qui comptent le nombre de sommets dans le stable courant et dans le graphe induit courant.

Question 5. Implementez cette nouvelle condition d'arret comme cas terminal de la recursion.

1.4 Format du fichier de sortie

Le resultat recherche par l'algorithme est un stable de taille maximum. C'est pour cela que vous devez retenir, au cours des appels recursifs a `calcule_stable_max`, la taille maximum des stables rencontres et la liste des sommets d'un stable ayant cette taille.
A la fin du programme, vous ecrirez ces informations dans le flux de sortie (`fout`) au format suivant :

- la premiere ligne contiendra un entier qui est la taille du stable maximum et
- la deuxieme ligne contiendra une suite d'entier, separes par un espace, qui sont les identifiants des sommets composant un stable maximum.

2 Application de l'algorithme

Appliquez votre algorithme sur les donnees fournies pour le TP1, en commencant par les graphes jouets puis en poursuivant par les petits reseaux en premier. Mesurez le temps d'execution de votre programme a l'aide de la commande `time COMMANDE` du systeme d'exploitation qui permet de savoir quel temps a pris l'execution de `COMMANDE`. Grace aux resultats sur les petits reseaux, essayez d'estimer le temps d'execution attendu pour les reseaux plus grands, en vous rappelant que la complexite theorique de l'algorithme que vous avez implemente est $O(1.45^n)$.

Question 6. En pratique, pour les donnees fournies, jusqu'a quelle taille pouvez vous traiter un reseau en un temps court, disons 5 minutes ?

Question 7. Y a-t-il une difference dans les temps d'execution observes pour les reseaux issus de donnees reelles et pour les reseaux synthetiques ?

3 Lister tous les stables maximum

Question 8. Par rapport au TP1, le programme prend une option de plus. A quoi sert-elle ?

Pour rappel, l'algorithme vu en cours, que vous venez d'implementer, permet non seulement de trouver un stable de taille maximum mais peut meme lister tous les stables de taille maximum. On souhaite maintenant implementer cette fonctionnalite dans le programme.
Le nombre de stables maximums dans un graphe peut etre tres grand, exponentiel en le nombre de sommets du graphe (jusqu'a $3^{\frac{n}{3}}$). Cela peut vite devenir trop volumineux pour etre stocke en memoire. Aussi, pour eviter de saturer la memoire, vous ecrirez sur le flux de sortie (`fout`) chaque stable maximum que vous trouverez au cours de l'algorithme. Dans certains cas, lorsque le nombre de stables maximums est trop important, il pourra etre utile de ne pas diriger le flux `fout` vers un fichier mais plutot vers la sortie standard (valeur par defaut lorsque vous ne specifiez pas l'option `-o` lors de l'appel au programme), qui s'affiche sur la console.
Si on ne veut pas stocker en memoire les stables rencontres, il faut connaitre a priori la taille maximum d'un stable. Vous procederez donc en deux passes. Dans la premiere

passe, vous utiliserez simplement la fonction `calcule_stable_max` que vous avez implementee pour calculer la taille maximum α d'un stable dans le graphe donne en entree a l'algorithme. Dans la deuxieme passe, vous rappelerez cette meme fonction, mais en mode *enumeration*, et vous ecrirez sur la sortie standard tous les stables de taille α que vous rencontrerez.
Concretement, on ne souhaite pas ecrire deux versions de la fonction `calcule_stable_max`. Cela signifie que vous ajouterez un parametre booleen, nomme `mode`, a `calcule_stable_max` pour savoir si elle est appelee en mode *classique* ou en mode *enumeration*.

Question 9. Modifiez votre fonction `calcule_stable_max` pour qu'elle prenne un parametre `mode` supplementaire et que lorsque ce parametre effectif est vrai, la fonction ecrive sur la sortie standard tous les stables maximum rencontres. Puis modifier votre programme principal pour qu'il liste tous les stables maximums lorsque l'option correspondante a ete activee lors de l'appel au programme.

Essayez votre nouvelle fonction sur les graphes jouets.

Question 10. Chaque stable maximum apparait-il une seule fois sur la sortie ? Pourquoi ? Peut on regler ce probleme ?