TP1 - Algorithme probabiliste pour la coupe minimum

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, en language C, l'algorithme probabiliste pour la coupe minimum vu en cours. 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 (avec de sections predefinies, en commentaire, qui vous indiquent ou placer certaines des parties de code que vous allez developpez) et le fichier `auxiliaire.c` contient des fonctions que vous utiliserez pour manipuler des graphes et leurs listes d'adjacences et tirer des nombres aleatoires. Vous coderez les fonctions necessaires a l'algorithme dans un troisieme fichier que vous creerez et que vous inclurez de la meme maniere au debut de `main.c`.
Le programme se compile grace a la commande `gcc -Wextra -Wall -O3 main.c -o mincut` et s'execute en tapant `./mincut`. Tapez `./mincut -h` pour afficher l'aide.

1 Prise en main du programme

Question 1. Compilez le programme. Combien d'options a-t-il ? A quoi servent-elles ? Quelles sont les valeurs par defaut ?

Question 2. A quoi sert la fonction `graph_from_file` du fichier `auxiliaire.c` ? Quelle autre fonction de manipulation de graphe ce fichier contient-il ?

Question 3. Comment est represente un graphe ? De quel type de graphe s'agit-il ?

Question 4. Quelle structure stocke une liste d'adjacence ? De quel type de liste s'agit-il ?

Question 5. Dans le fichier `auxiliaire.c`, ecrivez une fonction `free_nodl` qui libere entierement l'espace memoire occupe par une liste de noeuds, donnee par un pointeur vers la liste. Attention, il faut liberer l'espace occupe par chacune des cellules de la liste !
De meme, ecrivez une fonction `free_graph` qui libere entierement l'espace memoire occupe par un graphe, la encore donne par un pointeur sur le graphe.

> **Indication.** Avant de vous lancer, lisez l'annexe A sur la gestion de la memoire en C.

Pour l'instant, le programme se contente d'ouvrir les fichiers d'entree et de sortie specifies lors de l'appel au programme, respectivement en lecture seule et ecriture seule, et de les fermer a la fin du programme.

Question 6. Completez le `main` pour que le programme :
— charge un graphe G en memoire a partir du fichier specifie en entree (ou de l'entree standard),
— ecrive sur la sortie d'erreur standard (`stderr`) le nombre de sommets et le nombre d'aretes du graphe charge,
— ecrive le graphe charge dans le fichier specifie en sortie (ou sur la sortie standard) et
— libere la place prise par le graphe en memoire avant de quitter.
Vous fermerez le fichier d'entree immediatement apres avoir charge le graphe qu'il contient, a l'aide de la fonction `fclose`, deja utilisee a la fin du main.

> **Indication.** Vous trouverez de nombreux exemples d'ecriture dans un fichier ou sur les sorties standard dans les fichiers `main.c` et `auxiliaire.c` qui vous sont fournis. L'annexe B du sujet contient egalement un petit rappel sur l'utilisation de la fonction `fprintf`.

Plusieurs fichiers qui contiennent chacun un graphe, et dont le nom est de la forme `graphEL_*`, vous sont donnes dans le dossier `/data` accessible depuis la page web des TP (voir descriptif succint de leur provenance dans la section 3). Le format qu'ils utilisent pour encoder le graphe G est le suivant :
— la premiere ligne du fichier contient le nombre n de sommets dans le graphe,
— toutes les autres lignes du fichier contiennent une arete du graphe sous la forme de deux entiers u v separes par un espace, ou u et v sont les identifiants des deux sommets (distincts) qui sont les extremites de l'arete,
— l'identifiant d'un sommet est un entier compris entre 0 et $n-1$.

Question 7. Essayez votre programme sur quelques-uns des graphes fournis, en particulier sur `toygraph` et `toygraph2`. Y a-t-il des differences entre le fichier d'entree et le fichier de sortie ecrit par la fonction `write_graph` ? D'ou viennet ces differences ?

> **Indication.** Si besoin (pour les gros fichiers), vous pouvez comparez le contenu des deux fichiers de maniere automatique grace a la commande `diff fic1 fic2` du systeme d'exploitaion.

Question 8. Avant de continuer, dans le main, gardez la partie du code qui charge le graphe en memoire et retirez celle qui l'ecrit dans le fichier de sortie (dans la suite, on va s'occuper d'y ecrire le resultat de l'algorithme).

2 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`.

2.1 Contracter une arete

Le coeur de l'algorithme vu en cours est la fonction de contraction d'une arete. Une des implementations les plus simples de cet algorithme consiste a simuler les contractions d'aretes sans les effectuer reellement sur le graphe, c'est a dire que le graphe reste inchange durant tout l'algorithme. Pour cela on attribue a chaque sommet l'identifiant du groupe auquel il appartient, qui est toujours choisi comme l'identifiant d'un sommet du groupe. Initialement chaque sommet est dans un groupe different (dont l'identifiant est donc l'identifiant du sommet lui meme) et lors d'une contraction d'arete, on se contente de faire l'union des groupes des deux sommets qui sont les extremites de l'arete contractee. A la fin de l'algorithme, lorsqu'il ne reste que deux groupes, on obtient la valeur de la coupe correspondante en comptant le nombre d'aretes entre deux sommets qui n'appartiennent pas au meme groupe.
En terme de structure de donnee, on representera les groupes a l'aide de trois structures :

- un tableau de taille n qui a chaque sommet associe l'identifiant de son groupe et
- un tableau de taille n qui a chaque identifiant de groupe associe la liste des sommets qu'il contient et
- un tableau de taille n qui a chaque identifiant de groupe associe le nombre de sommets dans le groupe.

Question 9. Dans la section *DATA STRUCTURE* du `main`, declarez les trois tableaux representant les groupes, reservez l'espace memoire necessaire pour chacun d'eux et initialisez les comme ils doivent l'etre au debut de l'algorithme.

> **Indication.** Avant de vous lancer, lisez (ou relisez, si besoin) l'annexe A sur la gestion de la memoire en C.

Pour des questions de complexite, lors d'une contraction, on changera seulement les identifiants des sommets du plus petit groupe. Le probleme d'union des groupes auquel on se retrouve confronte est un probleme tres classique en informatique connu sous le nom d'*union-find*.

Question 10. Vous avez du deja voir le probleme d'union-find au cours de vos etudes. Quelle est la complexite totale de l'implementation proposee ici pour l'ensemble des unions realisees au cours de l'algorithme de contraction ?

Question 11.
Dans le fichier `fonctions-algo.c`, implementez la fonction `contraction_simulee` qui actualise, lors d'une contraction d'arete, les trois tableaux representant les groupes. Incorporez le contenu du fichier `fonctions-algo.c` a l'aide d'un `#include "fonctions-algo.c"` au debut du fichier `main.c`.

Question 12. Dans la section *COMPUTE MIN CUT* du `main`, testez votre fonction, et verifiez que son resultat est correct, pour la contraction d'une seule arete, puis pour quelques contractions successives. Faites cela avec l'un des deux graphes jouets fournis dans le dossier `/data` accessible depuis la page web des TP.

> **Indication.** Lors des contractions successives, prenez garde a ne contractez que des aretes qui existent encore dans le multigraphe contracte courant, c'est a dire qui sont entre des sommets appartenent a des groupes differents.

2.2 Tirer une arete au hasard

A chaque etape de l'algorithme, on doit tirer uniformement aleatoirement une arete parmi les aretes du multigraphe $\tilde{G}$ courant, que nous avons choisi de ne pas explicitement representer. Une facon simple et efficace de faire ces tirages aleatoires est de choisir d'emblee un ordre aleatoire π sur toutes les m aretes du graphe G de depart. Au cours de l'algorithme on parcourt cet ordre en considerant les aretes une par une : si l'arete consideree est une arete du multigraphe $\tilde{G}$ courant, c'est celle-ci que l'on contracte a cette etape de l'algorithme, sinon, si l'arete consideree a disparu du multigraphe dans les operations de contraction effectuees precedemment dans l'algorithme, on la rejette et on passe a l'arete suivante dans l'ordre π.

Question 13. Cette facon de proceder garantie-t-elle que l'arete tiree a une etape de l'algorithme est choisie uniformement aleatoirement parmi les aretes restantes dans le multigraphe $\tilde{G}$?

Question 14. Comment determiner si une arete uv de G appartient au multigraphe contracte $\tilde{G}$ courant, a l'aide des groupes auquels appartiennent les sommets u et v ?

Pour stocker un ordre sur les aretes, deux choix naturels sont de :

— soit utiliser un tableau de structures de type `arete` (un couple de sommets)

— soit utiliser deux tableaux de sommets, un contenant la premiere extremite de l'arete, l'autre la deuxieme extremite.

Question 15. En utilisant une des trois fonctions de tirage aleatoire contenu dans le fichier `auxiliaire.c`, ecrivez dans le fichier `fonctions-algo.c`, une fonction qui genere un ordre aleatoire des aretes du graphe G de depart.

Indication. Vous pourrez procedez comme suit. Commencez par stocker les aretes du graphe dans le (ou les) tableau resultat dans un ordre arbitraire, par exemple celui dans lequel vous rencontrez les aretes en parcourant les listes d'adjacences du graphe G. Faites attention a ne stocker chaque arete qu'une seule fois (elles sont presentes deux fois dans les listes d'adjacence). Ensuite, melangez aleatoirement le tableau en choisissant uniformenent aleatoirement a chaque etape i, pour i allant de 0 a $m-1$, l'arete que vous placez en position i du tableau, parmi les aretes qui sont dans les positions i a $m-1$ (c'est a dire celles que vous n'avez pas encore tirees). Pensez a placer l'arete qui etait en position i precedemment a la place de celle que vous avez tiree entre les positions i et $m-1$.

Indication. Afin de choisir laquelle des trois fonctions de tirage aleatoire fournies vous utiliserez, faite afficher les valeurs de `RAND_MAX`, `UINT_MAX` et `ULONG_MAX` **sur la machine sur laquelle vous travaillez**. Vous vous limiterez a traiter des graphes ayant au plus un milliard d'aretes.

2.3 Iteration de l'algorithme de contraction

Question 16. Ecrivez la boucle qui contracte une par une les aretes du graphe. Combien de fois cette boucle doit elle s'executer ?

Question 17. Dans le fichier `fonctions-algo.c`, ecrivez une fonction `valeur_coupe` qui calcule la valeur d'une coupe, donnee par les tableaux representant la coupe.

Pour avoir une bonne probabilite de trouver la coupe minimum, il faut appliquer l'agorithme de contraction aleatoire plusieurs fois, avec des tirages aleatoires d'aretes different.

Question 18. Ecrivez la boucle qui itere cet algorithme autant de fois que voulu, avec un ordre aleatoire sur les aretes different a chaque fois.

Question 19. Ajoutez le code permettant de retenir la plus petite coupe trouvee lors des iterations de l'algorithme probabiliste de contraction.

En fin de programme, on veut ecrire dans un fichier resultat la coupe minimum trouvee. On adoptera le format suivant : la premiere ligne du fichier contiendra la valeur de cette coupe, la deuxieme ligne la liste des identifiants des sommets qui sont dans la premiere partie de la coupe, separes par un espace, et la troisieme ligne la liste des identifiants des sommets qui sont dans la deuxieme partie de la coupe.

Question 20. Ecrivez a la fin du main, le code necessaire pour ecrire les resultats dans le fichier de sortie passe en parametre du programme par l'utilisateur.

3 Application de l'algorithme sur des jeux de donnees

Plusieurs jeux de donnees synthetiques ou provenant de contextes reels sont donnes dans le dossier `/data` accessible depuis la page web des TP. L'annexe C en contient une breve description.

Question 21. Pour chacun des graphes fournis, verifiez le nombre de sommets ecrit dans le fichier et comptez le nombre d'aretes presentes dans le fichier.

Indication. Vous vous aiderez de la commande `head -1 fic` du systeme d'exploitation, qui affiche la premiere ligne du fichier `fic` et de la commande `wc -l fic` qui affiche le nombre de lignes dans le fichier `fic`.

Question 22. Appliquez l'algorithme, avec une unique iteration, sur chacun des graphes fournis. Quelle est la valeur de la coupe trouvee ? Dans quel cas peut-on etre sur quil s'agit de la coupe minimum ?

Question 23. D'apres le cours, comment choisir le nombre d'iterations N pour obtenir la coupe minimum avec une probabilite d'au moins $0,999$? Pour quels graphes peut-on faire ce choix tout en gardant un temps d'execution raisonnable (c'est a dire avoir la reponse avant ce soir minuit) ?

Indication. Utilisez la commande `time COMMANDE` du systeme d'exploitation qui donne le temps pris par l'execution de la commande `COMMANDE`. Faites cela pour chaque graphe en fixant le nombre d'iterations de l'algorithme a 10.

Question 24. En faisant un tres grand nombre d'iterations, c'est a dire en choisissant N comme indique a la question precedente, estimez la probabilite de tomber sur la coupe minimum en une seule iteration, pour chacun des graphes fournis. Vous considererez qu'au cours de ces N iterations, vous avez en effet decouvert la coupe minimum (verifiez en comparant au resultat obtenu par vos camarades).

Question 25. Pour les graphes qui sont trop volumineux pour choisir N comme demande a la question precedente, vous prendrez N le plus grand possible et en regardant les resultats obtenus vous discuterez de la validite de l'hypothese selon laquelle vous avez obtenu la coupe minimum.

A Rappel sur la gestion de la memoire en C

En C, on doit gerer soi-meme la memoire. On alloue de l'espace memoire sur le tas grace a la fonction `malloc` qui s'utilise ainsi :

```
TYPE* p;
p = (TYPE*) malloc(sizeof(TYPE));
```

La premiere ligne declare un pointeur `p` sur un type `TYPE`, la deuxieme alloue sur le tas l'espace necessaire pour stocker une variable de type `TYPE`. L'argument de `malloc` est la taille en octet de la zone memoire allouee, le forcage de type `(TYPE*)` devant malloc est necessaire pour la concordance des types avec la variable `p` de type pointeur sur `TYPE`. Si on veut reserver un tableau de type `TYPE` et de taille `TAILLE`, on ecrit :

```
p = (TYPE*)malloc(TAILLE*sizeof(TYPE));
```

On accede alors a la case d'indice `i` du tableau `p`, pour `i` entre `0` et `TAILLE-1`, par l'expression `p[i]`.
Pour eviter les problemes lors du deboggage, on protegera toutes les allocations memoires en faisant quitter le programme avec une message d'erreur a chaque fois qu'une allocation est ratee. Pour cela on utilisera la fonction `report_error` fournie dans le fichier `auxiliaire.c`, de la maniere suivante :

```
if ( (p = (TYPE*) malloc(TAILLE*sizeof(TYPE))) == NULL )
   report_error("malloc() error");
```

Vous avez de multiples exemples d'utilisation de `malloc` dans le fichier `auxiliaire.c`.
Pour desallouer la zone memoire pointee par le pointeur `p` on utilise la fonction `free` :

```
free(p);
p=NULL;
```

Il n'y a pas de garantie sur ce que vaut `p` apres la desallocation. C'est pour cela que par securite, il est conseille de mettre `p` a `NULL` apres desallocation pour eviter d'avoir des pointeurs vers des zones desallouees. Il n'y a pas non plus de garantie sur ce que contient la zone desallouee apres desallocation : elle peut etre reinitialisee ou laissee intacte (le plus courant) et dans tous les cas elle peut bien sur etre reservee plus tard dans le programme a un autre usage.
Les fuites memoires sont un vrai probleme, elles vous empecherons de traiter de grandes instances en entree de votre programme (ce qui est votre but). Il faut donc leur faire la chasse. Pour les eviter, vous desallouerez systematiquement toute memoire reservee, dans le programme principal (`main`) ou dans un sous-programme (fonction ou procedure), des lors que vous savez qu'elle ne servira plus (et dans tous les cas, au plus tard a la fin du `main`). Vous pouvez verifier que votre programme n'a pas de fuite memoire en scrutant la place qu'il utilise en memoire lors de son execution a l'aide de la fonction `top` du systeme d'exploitaion. Par exemple,

```
top -d 1 -u username -o %MEM
```

vous affiche les statistiques des programmes que vous (avec le nom d'utilisateur `username`) avez lance tries par ordre decroissant d'utilisation de la memoire (`-o %MEM`) et actualisees toutes les secondes (`-d 1`). Plus d'info avec `man top`.

B Rappel sur la fonction `fprintf`

La fonction `fprintf` permet d'ecrire des donnees selon differents format sur un flux de sortie. Un exemple d'utilisation simple est

```
fprintf(FLUX,"Texte quelconque");
```

qui ecrit la chaine de caractere `"Texte quelconque"` sur le flux de sortie `FLUX`, qui peut etre un fichier (type `FILE*`) ouvert en ecriture ou un flux de sortie standard du systeme d'exploitation, comme `stdout` (sortie standard) ou `stderr` (sortie d'erreur standard). Le flux est le premier parametre de `fprintf` et la chaine a ecrire le deuxieme parametre. Il peut y avoir plus de parametres dans l'appel a `fprintf`, places apres le flux et la chaine, qui servent a incorporer d'autres donnees dans la chaine de caractere en specifiant le format auquel elle doivent etre ecrite sur le flux a l'aide du caractere `%`. Un exemple de syntaxe est le suivant :

```
fprintf(stderr,"Il y a %d sommets dans le graphe et %d aretes",n,15);
```

Si `n` est une variable de type `int` qui vaut 8, cela ecrit sur la sortie d'erreur standard la chaine de caracteres

```
Il y a 8 sommets dans le graphe et 15 aretes
```

Le format d'ecriture des entiers `n` et `15` est specifie par le `%d` qui est le format d'ecriture decimale des `int`. Les parametres de `fprintf` qui suivent la chaine de caractere, ici `n` et `15`, sont mis en correspondance avec les indications de format contenues dans la chaine en utilisant l'ordre dans lequel apparaissent les parametres supplementaires passes a la fonction `fprintf` et l'ordre dans lequel apparaissent les indications de format dans la chaine de caractere passee en parametre a `fprintf`. Le nombre de parametres supplementaire doit donc etre identique au nombre d'indications de format contenues dans la chaine et il doit y avoir concordance entre les indications de format et les types des parametres supplementaires. Par exemple, `%u` est le format d'ecriture decimale des `unsigned int`, `%lu` celui des `long unsigned int` et `%x` est le format d'ecriture hexadecimale des `unsigned int`. On aurait par exemple pu appeler `fprintf` ainsi

```
fprintf(stderr,"Il y a %d sommets dans le graphe et %x aretes",n,15);
```

ce qui ecrit la chaine de caracteres

```
Il y a 8 sommets dans le graphe et f aretes
```

Les retours a la ligne dans la chaine de caractere sont codes par `\n`. Exemple :

```
fprintf(stderr,"Il y a %d sommets dans le graphe\net %d aretes",8,15);
```

produit l'ecriture

```
Il y a 8 sommets dans le graphe
et 15 aretes
```

Pour optimiser les performances du systeme, l'ecriture ne se fait pas directement sur le flux specifie en parametre de `fprintf` mais passe au prealable par un cache dont le contenu n'est effectivement ecrit sur le flux que lorsque le cache est suffisemment rempli. On peut forcer l'ecriture sur le flux et le vidage du cache a l'aide de la fonction `fflush` :

```
fflush(FLUX);
```

Afin de faciliter la phase de deboggage, il est conseiller de proceder au vidage force du cache regulierement, apres chaque etape importante d'ecriture. En effet, lorsque le programme quitte subrepticement, a cause d'un bug par exemple, le cache des flux ouverts en ecriture n'est pas vide. Ainsi, tout ce qui avait ete ecrit sur le flux mais qui etait encore dans le cache est perdu et n'apparait pas sur le flux. Si l'on se fie a l'affichage sur le flux, cela trompe sur le moment de l'execution auquel le bug s'est produit.

C Jeux de donnees

Vous trouverez plusieurs graphes au format decrit au debut du sujet dans le dossier `/data` accessible depuis la page web des TP. Certains de ces graphes sont synthetiques, c'est a dire qu'ils ont ete generes soit a la main soit par des methodes de generation automatiques, et d'autres proviennent d'operations de mesure realises dans des contextes concrets : Internet, publications scientifiques, biologie, environnement, reseaux pair-a-pair, reseaux routiers et reseaux sociaux en ligne. Une breve description de ces graphes est donnee ci-dessous.

`graphEL_toygraph`, 7 sommets, graphe jouet construit a la main,

`graphEL_toygraph2`, 8 sommets, graphe jouet construit a la main,

`graphEL_rand_100_8`, 100 sommets, graphe aleatoire (Erdos-Renyi G_{nm}) de degre moyen 8,

`graphEL_rand_500_16`, 500 sommets, graphe aleatoire (Erdos-Renyi G_{nm}) de degre moyen 16,

`graphEL_rand_1000_16`, 1000 sommets, graphe aleatoire (Erdos-Renyi G_{nm}) de degre moyen 16,

`graphEL_as2000`, 6474 sommets, graphe de connections entre les systemes autonomes d'Internet en 2000,

`graphEL_ca-GrQc`, 4158 sommets, graphe de coauteurs dans le domaine de la relativite generale et de la cosmologie quantique,

`graphEL_figeys`, 2217 sommets, graphe d'interactions chimiques entre proteines,

`graphEL_foodweb`, 183 sommets, graphe de predation entre especes (chaine alimentaire),

`graphEL_p2p-Gnutella`, 62561 sommets, graphe de connection entre pairs dans le reseau P2P Gnutella,

`graphEL_roadNet-TX`, 1351137 sommets, graphe du reseau routier du Texas,

`graphEL_youtube`, 1134890 sommets, une partie du reseau social de youtube.