Concours commun Mines-Ponts, France

Session 2024, filière MPI

Informatique 1

Durée : 3 heures

Ce que traite le sujet

- Section 1 : entourage de plus petite bordure
- Section 2 : algorithme de Goldberg
- Section 3 : algorithme de Charikar

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/mines/mpi/2024/info1

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

A2024 – INFO I MPI

ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.

Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).

CONCOURS 2024

PREMIÈRE ÉPREUVE D'INFORMATIQUE

Durée de l'épreuve : 3 heures

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve concerne uniquement las candidats de la filière MPI.

L'énoncé de cette épreuve comporte 9 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Les sujets sont la propriété du GIP CCMP. Ils sont publiés sous les termes de la licence
Creative Commons Attribution - Pas d'Utilisation Commerciale - Pas de Modification 3.0 France.

Tout autre usage est soumis à une autorisation préalable du Concours commun Mines Ponts.

Préliminaires

Présentation du sujet

L'épreuve est composée d'un problème unique comportant 33 questions divisées en trois sections. L'objectif du problème est d'extraire un *sous-graphe le plus dense* d'un graphe non orienté quelconque, c'est-à-dire repérer un sous-ensemble de sommets riche en arêtes. Le calcul d'un sous-graphe dense possède des applications considérables dans le domaine de la fouille des données : qu'il s'agisse de détection de communautés dans des réseaux sociaux, de repérage d'attaques par pollupostage (ou *spamming*) ou d'identification de complexes protéiques au sein de données biologiques massives pour ne citer que quelques exemples.

Dans la première section (page 1), nous résolvons un problème de minimisation de bordure dans un multigraphe orienté à l'aide d'une technique de *chemins augmentants*. Dans la deuxième section (page 5), nous étudions une technique de *réduction du problème* de construction d'un sous-graphe le plus dense au problème de la première section. Dans la troisième section (page 8), nous étudions un *algorithme d'approximation* plus rapide.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique avec la police en italique (par exemple n, $n_{\max}$ ou n_1) et du point de vue informatique avec celle en romain avec espacement fixe (par exemple `n`, `NMAX` ou `n1`).

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

Selon les consignes, il faudra coder des fonctions à l'aide du langage de programmation C exclusivement, en reprenant le prototype de fonction fourni par le sujet, ou en pseudo-code (c-à-d. dans une syntaxe souple mais conforme aux possibilités offertes par le langage C). Inclure les entêtes tels que `<assert.h>`, `<stdbool.h>`, etc. n'est pas demandé.

Quand l'énoncé demande de coder une fonction, sauf indication explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites.

Le barème tient compte de la clarté et de la concision des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle. Lorsqu'une réponse en pseudo-code est permise, seule la logique de programmation est évaluée, même dans le cas où un code en C a été fourni en guise de réponse.

1 Entourage de plus petite bordure

Dans cette section, nous travaillons dans un *multigraphe orienté*, c'est-à-dire un couple (W, F, μ) où W est un ensemble fini, appelé ensemble de sommets, F est une partie de $W \times W$, appelée ensemble d'arcs, et $\mu : F \to \mathbb{N}^*$ est une application, appelée multiplicité des arcs.

Les démonstrations ou justifications pourront utiliser la notion de *multi-ensemble* sans formalité excessive. L'intersection, notée $\cap$, se calcule en considérant le minimum des multiplicités des éléments ; l'union disjointe, notée $\sqcup$, se calcule en considérant la somme des multiplicités des éléments ; le cardinal se calcule en sommant toutes les multiplicités.

1.1 Réseaux

Définitions : Nous appelons *réseau* tout quintuplet $H = (W, F, \mu, s, t)$ tel que
- W est un ensemble fini, appelé ensemble de *sommets*,
- F est un sous-ensemble de couples de $W \times W$, appelé ensemble d'*arcs*,
- μ est une application $F \to \mathbb{N}^*$, l'entier naturel $\mu(e)$ étant appelé la *multiplicité* de l'arc e,
- s est un sommet particulier de W, appelé la *source*,
- t est un sommet particulier de W, appelé le *puits*.

Indication C : Par convention, nous numérotons les sommets d'un réseau $H = (W, F, \mu, s, t)$ par les entiers entre 0 et $n+1$, où $n+2$ est le cardinal de W. La source s porte le numéro 0 ; le puits t porte le numéro $n+1$.

Nous préparons deux fichiers, `network.h` et `network.c`. Le premier fichier contient les déclarations suivantes.

```
1.  /* Debut du fichier network.h */
2.
3.  #ifndef NETWORK_H
4.  #define NETWORK_H
5.
6.  const unsigned int NMAX = 102;
7.
8.  struct network_s {
9.      unsigned int n;
10.     unsigned mu[NMAX][NMAX];
11. };
12. typedef struct network_s network;
13.
14. void nw_init(unsigned int n);
15. void nw_add(int u, int v);
16. void nw_remove(int u, int v);
17. void nw_disconnect(void);
18.
19. void _arr_init(int *T);
20. bool _bfs_tree(void);
21. void _residue(void);
22.
23. #endif
24.
25. /* Fin du fichier network.h */
```

☐ 1 – Expliquer le rôle des lignes 3, 4 et 23 dans le fichier `network.h`.

Le second fichier contient initialement les définitions suivantes et sera complété par les fonctions de la section 1.

```
26.  /* Debut du fichier network.c */
27.
28.  network H;
29.  int A[NMAX];
30.
31.  /* Fin du fichier network.c */
```

Pour tout couple $(u, v) \in W \times W$, le champ `mu[u][v]` de la variable gobale H vaut 0 si (u, v) n'est pas un arc du réseau et vaut $\mu((u, v))$ si (u, v) est un arc.

☐ 2 – Expliquer comment faire interagir les fichiers `network.h` et `network.c`.

☐ 3 – Écrire une fonction C **`void`** `nw_init(`**`unsigned int`** `n)` ainsi spécifiée :
Précondition : L'entier naturel n est strictement inférieur à la constante globale $n_{\max}$.
Effet : La précondition est vérifiée par une assertion.
Postcondition : La variable globale H contient le réseau vide avec $n + 2$ sommets et aucun arc.

☐ 4 – Écrire une fonction C **`void`** `nw_add(`**`int`** `u, `**`int`** `v)` qui ajoute une occurence de l'arc (u, v) au réseau (W, F, μ, s, t) stocké dans la variable globale H, autrement dit, qui incrémente la multiplicité $\mu[u][v]$ d'une unité.

☐ 5 – Écrire une fonction C **`void`** `nw_remove(`**`int`** `u, `**`int`** `v)` qui retire une occurence de l'arc (u, v) dans le réseau stocké dans la variable globale H et qui se défend, par le truchement d'une assertion, du cas où l'arc à supprimer n'existe pas.

☐ 6 – Écrire une fonction C **`void`** `_arr_init(`**`int`** `*T)` ainsi spécifiée :
Précondition : Le pointeur T désigne un tableau d'entiers formé de $n + 2$ cases, où $n + 2$ est le nombre de sommets du réseau stocké dans la variable globale H.
Postcondition : Toutes les cases du tableau désigné par le pointeur T valent -1.

Nous utilisons le terme *chemin* pour signifier systématiquement un chemin simple dans le graphe orienté (W, F), c'est-à-dire une suite d'arcs $(e_1, \ldots, e_t)$ de F distincts qui se succèdent. Le sommet initial d'un chemin s'appelle la *source*, le sommet final s'appelle le *puits*.

☐ 7 – Écrire une fonction C **`bool`** `_bfs_tree(`**`void`**`)` ainsi spécifiée :
Précondition : La variable globale H contient un réseau (W, F, μ, s, t) à $n + 2$ sommets.
Effet : La fonction calcule une arborescence $\mathcal{A}$ de plus courts chemins selon un parcours en largeur dans le graphe (W, F) et l'écrit dans la variable globale A.
Postcondition : On a $A[s] = s$ et, pour tout autre sommet $v \in W$ accessible depuis s, la case $A[v]$ contient le prédécesseur du sommet v dans l'arborescence $\mathcal{A}$. Enfin, toute autre case de A vaut -1.
Valeur de retour : Booléen `true` si le puits t est accessible depuis la source s dans l'arborescence $\mathcal{A}$. Booléen `false` sinon.

1.2 Déconnexion de la source et du puits

Définition : Nous disons qu'un ensemble de chemins $\Pi = \{\pi_1, \ldots, \pi_k\}$ de source commune s et de puits commun t est *réalisable* dans le réseau $H = (W, F, \mu, s, t)$ si, pour tout arc $e \in F$, les chemins de Π utilisent moins de $\mu(e)$ fois l'arc e, autrement dit

$$\forall e \in F, \quad \text{Card}\left(\{j \in [\![1, k]\!] \text{ tel que } e \in \pi_j\}\right) \leqslant \mu(e).$$

Définition : Étant donné un ensemble réalisable de chemins $\Pi = \{\pi_1, \ldots, \pi_k\}$ de source commune s et puits commun t dans un réseau $H = (W, F, \mu, s, t)$, nous appelons *réseau résiduel de H par rapport à Π*, et nous notons $\text{Res}(H, \Pi)$, le réseau obtenu à partir de H en effectuant, pour tout indice j compris entre 1 et k et pour tout arc $e = (u, v) \in \pi_j$, la suppression d'une occurence de l'arc e et l'ajout d'une occurence de l'arc opposé $\overline{e} = (v, u)$.

☐ 8 – Écrire une fonction C `void _residue(void)` ainsi spécifiée :
Précondition : La variable globale H contient un réseau (W, F, μ, s, t). L'ensemble des arcs $\{(A[v], v); 0 < v \leqslant n + 1 \text{ avec } A[v] > 0\}$ tiré de la variable globale A forme une arborescence $\mathcal{A}$ de racine $s = 0$ atteignant le puits t dans le graphe (W, F).
Postcondition : La variable H contient le réseau résiduel $\text{Res}\left((W, F, \mu, s, t), \{\alpha\}\right)$ où α est l'unique chemin de source s et de puits t dans l'arborescence $\mathcal{A}$.

☐ 9 – Nous appliquons k fois la fonction `_residue` à un réseau H en mettant à jour la variable globale A et notons α_1, …, α_k les chemins utilisés entre la source s et le puits t. Montrer qu'il est toujours possible de trouver un ensemble réalisable Π de k chemins dans H tels que le réseau obtenu des applications de `_residue` coïncide avec le réseau résiduel par rapport à Π, autrement dit tel que l'on a

$$\text{Res}\left(\ldots \text{Res}\left(\text{Res}\left(H, \{\alpha_1\}\right), \{\alpha_2\}\right), \ldots, \{\alpha_k\}\right) = \text{Res}(H, \Pi).$$

☐ 10 – Écrire une fonction C `void nw_disconnect(void)` ainsi spécifiée :
Précondition : La variable globale H contient un réseau (W, F, μ, s, t).
Effet : Tant qu'il existe un chemin de source s et de puits t, le réseau H est transformé en le réseau résiduel $\text{Res}((W, F, \mu, s, t), \{\alpha\})$, où α est un plus court chemin de s vers t.
Postcondition : Si la fonction se termine, il n'existe pas de chemin de la source vers le puits dans le réseau H. De plus, l'ensemble des arcs $\{(A[v], v); 0 \leqslant v \leqslant n + 1 \text{ avec } A[v] > 0\}$ tiré de la variable globale A forme une arborescence $\mathcal{A}$ dans le réseau résiduel dans laquelle la racine est s et le sommet t n'est pas accessible.

☐ 11 – Démontrer que la fonction `nw_disconnect` se termine en introduisant un variant de boucle inspiré par la question 9.

1.3 Optimalité

Définition : Soit $H = (W, F, \mu, s, t)$ un réseau et S un sous-ensemble de sommets de W. Nous appelons *bordure*, et notons ∂S, l'ensemble d'arcs allant de S vers son complémentaire :

$$\partial S = \{(u, v) \in F \text{ avec } u \in S \text{ et } v \notin S\}.$$

Nous étendons la notion de multiplicité à des ensembles d'arcs en posant :

$$\mu(\partial S) = \sum_{e \in \partial S} \mu(e).$$

Définition : Nous appelons *entourage* dans le réseau $H = (W, F, \mu, s, t)$ tout sous-ensemble de sommets $S \subseteq W$ contenant la source s mais pas le puits t.

☐ 12 – Soient Π un ensemble réalisable de k chemins dans un réseau $H = (W, F, \mu, s, t)$ et $S \subseteq W$ un entourage dans H. Établir la relation

$$k \leqslant \mu(\partial S).$$

Nous fixons pour les quatre prochaines questions un ensemble réalisable Π_0 de k_0 chemins dans un réseau H tel que, dans le réseau résiduel $\mathrm{Res}(H, \Pi_0)$, il n'existe plus aucun chemin entre la source s et le puits t. Nous notons S_0 l'entourage dans H composé des sommets accessibles dans $\mathrm{Res}(H, \Pi_0)$ depuis la source s.

☐ 13 – Montrer que, pour tout arc $e = (u, v) \in F$ tel que $u \in S_0$ et $v \notin S_0$, toutes les $\mu(e)$ occurrences de l'arc e sont utilisées par un chemin de l'ensemble Π_0.

☐ 14 – Montrer qu'aucune occurence d'un arc $(v, u) \in F$ tel que $v \notin S_0$ et $u \in S_0$ n'est utilisée par un chemin de l'ensemble Π_0.

☐ 15 – Démontrer l'égalité

$$k_0 = \mu\left(\partial S_0\right).$$

Définition : Nous disons qu'un entourage S est *de plus petite bordure* s'il minimise la multiplicité $\mu(\partial S)$. Nous notons $\sigma(H)$ la multiplicité de la bordure d'un tel entourage.

☐ 16 – Démontrer la relation

$$k_0 = \sigma(H)$$

et en déduire que, si la variable globale H contient un réseau, après exécution de la fonction `nw_disconnect`, l'ensemble $\{v \in W\,;\ A[v] \geqslant 0\}$ est un entourage de plus petite bordure de H, à savoir un entourage de multiplicité $\sigma(H)$.

2 Algorithme de Goldberg

2.1 Densité d'un graphe

Pour tout ensemble fini X, la notation $\binom{X}{2}$ désigne l'ensemble des paires (non ordonnées) à valeur dans X, c'est-à-dire l'ensemble

$$\binom{X}{2} = \left\{\{u, v\} \text{ avec } (u, v) \in X^2 \text{ tel que } u \neq v\right\}.$$

Définition : Un *graphe non orienté* est un couple $G = (V, E)$ où V est un ensemble fini, appelé ensemble de sommets, et $E \subseteq \binom{V}{2}$ est un ensemble de paires, appelé ensemble d'arêtes. Pour tout ensemble de sommets $X \subseteq V$, nous notons $G_X = (X, E_X)$ le graphe $G_X = \left(X, E \cap \binom{X}{2}\right)$, appelé *graphe induit par X sur G*.

Indication C : Par convention, nous numérotons les sommets de tout graphe non orienté $G = (V, E)$ par les entiers entre 1 et n, où n est le cardinal de V. Nous représentons un graphe par un tableau de listes d'adjacence doublement chaînées en adoptant les déclarations de type suivantes.

```
32.  struct link_s {
33.      int node;
34.      struct link_s *prev;
35.      struct link_s *next;
36.  };
37.  typedef struct link_s link;
38.
39.  struct graph_s {
40.      unsigned int n;
41.      link **neighbours;
42.  };
43.  typedef struct graph_s graph;
```

Le champ `neighbours` est un tableau alloué dynamiquement de longueur $n + 1$ et dont la première case n'est pas utilisée. Chaque arête est stockée sous forme de deux arcs.

☐ 17 – Écrire une fonction C `graph *gr_create(unsigned int n)` dont la valeur de retour est un graphe non orienté vide à n sommets.

☐ 18 – Écrire une fonction C `void gr_add(graph *G, int u, int v)` ayant pour effet d'insérer une arête $\{u, v\}$ dans le graphe non orienté G.

Définition : Nous notons $\deg_G(v)$ le *degré* d'un sommet v dans le graphe non orienté $G = (V, E)$:

$$\deg_G(v) = |\{u \in V \text{ tel que } \{u, v\} \in E\}| = \sum_{\substack{u \in V \text{ t.q.} \\ \{v,u\} \in E}} 1.$$

☐ 19 – Écrire une fonction C `int degree(graph *G, int v)` dont la valeur de retour est le degré $\deg_G(v)$ du sommet v dans le graphe non orienté G.

☐ 20 – Écrire une fonction C `int edges(graph *G)` dont la valeur de retour est le nombre d'arêtes dans le graphe non orienté G.

Définition : Nous appelons densité du graphe $G = (V, E)$ le rapport

$$\delta(G) = \frac{m}{n}$$

où n est le cardinal de V et m est le cardinal de E.

☐ 21 – Écrire une fonction C `double density(graph *G)` dont la valeur de retour est la densité $\delta(G)$ du graphe non orienté G.

Définition : Le problème d'optimisation du *sous-graphe le plus dense d'un graphe non orienté* G consiste à trouver un ensemble de sommets non vide X tels que la densité $\delta(G_X)$ du sous-graphe induit $G_X = (X, E_X)$ est maximale et calculer cette densité maximale

$$\rho(G) = \max_{\substack{X \subseteq V; \\ X \neq \varnothing}} \delta(G_X).$$

2.2 Oracle pour le problème de décision

Nous nous intéressons au problème de décision suivant :

> Étant donnés un graphe non orienté G à n sommets et un entier naturel r, existe-t-il un sous-graphe induit $G_X = (X, E_X)$ de densité $\delta(G_X)$ strictement supérieure au seuil $\dfrac{r}{n^2}$? Autrement dit, a-t-on $\rho(G) > \dfrac{r}{n^2}$?

Définition : Étant donnés un graphe non orienté $G = (V, E)$ avec n sommets et m arêtes ainsi qu'un entier naturel r, nous appelons *réseau associé au graphe G et au seuil r* le réseau $H = (W, E, \mu, s, t)$ où

(α) L'ensemble W est la réunion $W = V \cup \{s, t\}$, les sommets source s et puits t étant de nouveaux sommets n'appartenant pas à V.

(β) L'ensemble d'arcs F comprend :
- pour tout sommet $v \in V$, un arc (s, v), de multiplicité mn^2.
- pour toute arête $\{u, v\} \in E$, un arc (u, v) et un arc (v, u), de multiplicité n^2.
- pour tout sommet $u \in V$, un arc (u, t), de multiplicité $mn^2 - n^2 \deg_G(u) + 2r$.

☐ 22 – Écrire une fonction C `void reduce(graph *G, int r)` ayant pour effet d'initialiser la constante globale H, de type `network`, par le réseau associé au graphe G et au seuil r.

☐ 23 – Soient $G = (V, E)$ un graphe non orienté et $X \subseteq V$ une partie non vide des sommets. Montrer que la densité du graphe induit $G_X = (X, E_X)$ satisfait la relation

$$\delta(G_X) = \frac{\displaystyle\sum_{v \in X} \deg_G(v) - \sum_{\substack{u \in X, v \in V \setminus X \\ \text{t.q.} \{u,v\} \in E}} 1}{2|X|}.$$

☐ 24 – Soient $G = (V, E)$ un graphe non orienté avec n sommets et m arêtes, $X \subseteq V$ une partie non vide des sommets, H le réseau associé au graphe G et à un seuil r et S l'entourage $S = \{s\} \cup X$ dans H. Montrer que la multiplicité de la bordure de S satisfait l'égalité

$$\mu(\partial S) = mn^3 + 2|X| \left(r - n^2\, \delta(G_X)\right)$$

et que

$$\mu(\partial\{s\}) = mn^3.$$

☐ 25 – Soit H le réseau associé au graphe G et au seuil r et soit $\sigma(H)$ la multiplicité de la bordure d'un entourage de plus petite bordure. Montrer les équivalences suivantes :

$$\begin{cases} \sigma(H) = mn^3 & \Leftrightarrow \quad \rho(G) \leqslant \dfrac{r}{n^2}, \\ \sigma(H) < mn^3 & \Leftrightarrow \quad \rho(G) > \dfrac{r}{n^2}. \end{cases}$$

☐ 26 – Écrire une fonction C `bool oracle(graph *G, int r)` dont la valeur de retour est le booléen `true` si $\rho(G) > \dfrac{r}{n^2}$ et `false` si $\rho(G) \leqslant \dfrac{r}{n^2}$. En justifier le principe. On pourra s'intéresser au contenu du tableau global A après exécution de la fonction `nw_disconnect`.

2.3 Recherche dichotomique

Nous considérons l'ensemble de couples d'entiers

$$\Delta = \left\{(m', n') \in \mathbb{N}^2 \text{ avec } 0 \leqslant m' \leqslant m \text{ et } 1 \leqslant n' \leqslant n\right\}.$$

☐ 27 – Soient deux couples (m_1, n_1) et (m_2, n_2) appartenant à Δ. Montrer que,

$$\text{si } \frac{m_1}{n_1} \neq \frac{m_2}{n_2}, \quad \text{alors} \quad \left|\frac{m_1}{n_1} - \frac{m_2}{n_2}\right| \geqslant \frac{1}{n^2}.$$

☐ 28 – Soient $G = (V, E)$ un graphe non orienté à n sommets et X une partie non vide de V. Montrer que s'il n'existe aucun sous-graphe induit de densité supérieure ou égale à $\delta(G_X) + \dfrac{1}{n^2}$, alors le sous-graphe induit G_X est un sous-graphe de densité maximale $\rho(G)$.

☐ 29 – Écrire une fonction C efficace `void binary_search(graph *G)` qui, pour tout graphe $G = (V, E)$, modifie le tableau global A de sorte que l'ensemble de sommets $X = \{v \in V\,;\ A[v] \geqslant 0\}$ vérifie

$$\delta(G_X) = \rho(G).$$

Justifier brièvement le principe de fonctionnement de la fonction.

☐ 30 – Nous supposons avoir écrit l'ensemble des fonctions de la section 2 dans un fichier `densest_subgraph.c`. Expliquer comment faire interagir le fichier `densest_subgraph.c` avec les fichiers `network.h` et `network.c` de la section 1.

3 Algorithme de Charikar

Dans le cadre d'un compromis entre temps d'exécution et justesse du calcul d'optimisation, nous nous intéressons à l'algorithme suivant.

Algorithme 1 : Recherche gloutonne d'une densité élevée

Entrées : Graphe $G = (V, E)$.
Sorties : Sous-graphe induit à grande densité

```
1 Γ ← G ;                          /* Plus dense graphe rencontré */
2 G_n ← G où n = |V|
3 pour i de n à 2 (cas inclus) faire
4     si δ(G_i) > δ(Γ) alors
5         Γ ← G_i
6     Identifier le sommet v_i qui minimise le degré deg_{G_i}.
7     Former le graphe G_{i-1} en supprimant de G_i le sommet v_i et les
        arêtes incidentes à v_i.
8 retourner Γ
```

3.1 Analyse d'un facteur d'approximation

Définition : Nous appelons *orientation* d'un graphe non orienté $G = (V, E)$ tout graphe orienté $\widehat{G} = (V, \widehat{E})$ tel qu'il existe une bijection $\omega : E \to \widehat{E}$ avec $\omega(\{u, v\}) = (u, v)$ pour tout arc $(u, v) \in \widehat{E}$. Le *degré entrant* d'un sommet v dans un graphe orienté $\widehat{G} = (V, \widehat{E})$ est

$$\deg^-_{\widehat{G}}(v) = \mathrm{Card}\left(\left\{u \in V \text{ tel que } (u, v) \in \widehat{E}\right\}\right) = \sum_{\substack{u \in V \text{ t.q.} \\ (u,v) \in \widehat{E}}} 1.$$

☐ 31 – Établir la majoration suivante entre la densité d'un graphe non orienté quelconque G et le degré entrant maximum d'une orientation quelconque $\widehat{G}$ de G

$$\delta(G) \leqslant \max_{v \in V} \deg^-_{\widehat{G}}(v).$$

Nous exécutons l'algorithme 1 sur un certain graphe non orienté G. Dans les questions qui suivent, nous numérotons les sommets de G selon la même numérotation que celle de l'algorithme. Nous notons ω_0 la bijection qui oriente l'arête $\{v_i, v_{i'}\}$ dans le sens $(v_{\min(i,i')}, v_{\max(i,i')})$ et $\widehat{G}_0$ l'orientation associée. Autrement dit, nous orientons toute arête, au moment de sa suppression, vers le sommet incident qui a conduit à sa suppression.

☐ 32 – Établir, pour tout i compris entre 1 et n, l'inégalité

$$\deg^-_{\widehat{G}_0}(v_i) \leqslant 2\delta(G_i)$$

et en déduire que l'algorithme 1 est un algorithme d'approximation dont on précisera le facteur d'approximation.

3.2 Implémentation optimale

☐ 33 – Détailler une structure de données qui permette d'exécuter l'algorithme 1 en temps $O(n + m)$ où $n = |V|$ et $m = |E|$. Il est suggéré de nantir la structure de données `graph` d'une table qui maintient, pour tout entier d compris entre 0 et n, la collection des sommets de degré d ainsi que d'autres éléments que l'on jugera opportuns.
Une réponse décrite en pseudo-code ou en langage naturel est admise mais doit être suffisamment précise pour s'assurer que la complexité est belle et bien linéaire.

Fin de l'épreuve

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MPI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Langage C : types, tableaux, pointeurs, allocation
- Spécifier, tester, prouver : terminaison et correction
- Listes et algorithmes de recherche (séquentielle, dichotomique)



Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière :
écrire sur WhatsApp

Les autres années et épreuves : prepahelp.com/annales/mines/mpi