L3-INFO Algorithmique et Modélisation

UFR IM²AG

UNIVERSITÉ Grenoble Alpes

Examen – 29 mai 2020 à 10h
durée 3 h de travail (difficultés techniques non comptées)

Informations préliminaires à l'examen, à lire impérativement

Dates

- disponibilité du sujet : 29 mai 2020 à 10h
- date limite de remise de la copie (dépot sur le moodle) : 30 mai 2020 à 10h

Diffusion du sujet

- diffusion par mail sur la liste officielle des étudiants de L3-INFO
- mise à disposition sur la plateforme moodle
  https://im2ag-moodle.e.ujf-grenoble.fr

Remise des copies

- Déposer le fichier sur la plateforme moodle de l'UFR cours Algorithmique et modélisation
  https://im2ag-moodle.e.ujf-grenoble.fr
- Le fichier doit être au format pdf
- la taille maximale de fichier est 20 Mo

Interaction de l'étudiant avec l'équipe enseignante

- **Hotline** le 29 mai 2020 de 10h à 13h, passé 13h nous ne répondrons qu'aux questions purement techniques ou d'organisation sans garantie de délai.
- Les interactions se feront par mail avec comme serveur de mail le serveur de l'UGA. utilisant l'adresse officielle de l'étudiant et le préfixe du sujet du mail « [L3-INFO :Algo6 :Examen] » le sujet devra être explicite pour permettre un traitement plus efficace.
- **Question concernant le sujet**
  uniquement adressée à l'équipe enseignante : Jean-Marc Vincent, Fanny Dufossé, Vincent Fagnon, copie pour information à Latifa Hamed-Abdelouahab
  Poser une question sur le sujet
- **Question concernant les aspects techniques**
  uniquement adressée à David Beniamine, Cyril Labbe et Jean-Marc Vincent copie pour information à Latifa Hamed-Abdelouahab
  Poser une question sur une difficulté technique (connexion au moodle, format de fichier, ...)
- **Question concernant l'organisation ou des difficultés personnelles**
  uniquement adressée à Jean-Marc Vincent et à Anne Rasse, éventuellement copie pour information à Latifa Hamed-Abdelouahab
  Poser une question personnelle
- la réponse sera soit individuelle, soit collective sur la liste mail des étudiants (précisions sur le sujet)
  les réponses ne seront peut-être pas instantanées.

Engagements de l'étudiant

Un examen à distance repose sur la confiance de l'équipe enseignante dans les étudiants. C'est pourquoi nous vous demandons de vous engager.

- Chaque étudiant s'engage à nous signaler au plus tôt toute difficulté d'organisation, technique ou personnelle rencontrée.
- Chaque étudiant s'engage à faire l'examen lui-même, sans aucune interaction (liée à l'examen) avec une autre personne durant toute la période d'examen, c'est à dire jusqu'à la date et l'heure maximale de remise des copies, c'est à dire le 30 mai 2020 à 10h exception faite uniquement avec des personnes responsables de l'épreuve et citées ci-dessus.
- Chaque étudiant s'engage à respecter la durée de l'examen, dans ce cas, la **durée de travail est estimée à 3h**. Nous rappelons que la règle des "24 heures" a été mise en place pour libérer les étudiants du stress engendré par des conditions de travail difficiles (mauvaise connexion, environnement bruyant, interruptions intempestives, contraintes,...)

Le respect des engagements ci-dessus sont basés sur la confiance, nous n'effectuerons pas de contrôle à distance du comportement des étudiants. Cependant les procédures standard contre les tentatives de fraude, comme la détection de plagiat, restent actives. En cas de suspicion ou de fait constaté de fraude, les règlements de l'UGA s'appliqueront.

D'autre part, nous vous demandons de faire attention à ne pas communiquer avec d'autres personnes au sujet de l'examen au moins 48h après la date de limite de remise le 30 mai 2020 à 10h, un mail vous informera que le debriefing de l'examen est possible.

Quelques conseils pour aider à la réussite de l'examen

i. tout fichier qui ne sera pas "PDF document" sera rejeté à la correction, la plateforme accepte les fichiers dont l'extension a été changée, donc ne vérifie pas le véritable format du fichier. Pour vérifier le format de votre fichier vous pouvez utiliser la commande UNIX `file` :

```
(base) my-machine$ file Nom_Prenom_L3_S6_Algo.pdf
  Nom_Prenom_L3_S6_Algo.pdf: PDF document ...
```

ii. Le fichier rendu doit contenir les réponses aux questions dans l'ordre dans lequel elles ont été posées. Si le format du fichier est contraint, vous pouvez composer votre copie à votre convenance : texte tapé (préférable), dessins et schémas explicatifs scannés (pour ne pas perdre trop de temps), les parties contenant des formules mathématiques peuvent être scannées. Le tout doit être parfaitement lisible par un outil de visualisation classique de pdf, vérifier que tout est parfaitement lisible avant de déposer votre copie.

iii. Les documents et calculette sont autorisés, ne pas perdre de temps à rechercher sur internet des solutions aux questions, les compétences évaluées dans cet examen sont principalement vos capacités d'analyse (compréhension) et de synthèse (restitution), ainsi que votre recul par rapport au domaine de l'algorithmique.

iv. Le barème est indicatif, il sera tenu compte de la qualité de la rédaction et de la clarté de la présentation.

v. En cas de difficulté de compréhension d'une question, il ne faut pas hésiter à faire des hypothèses complémentaires, expliquer comment vous avez compris la question et répondre. Vous êtes en autonomie face au sujet.

vi. Le sujet d'examen contient plusieurs problèmes, il faudra penser à organiser votre temps et ne pas se concentrer uniquement sur l'un d'entre eux.

Format de rendu

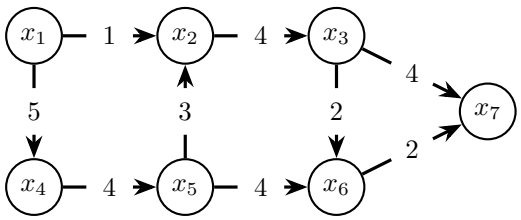
La copie d'examen doit être rendue au format pdf (pas de limitation du nombre de pages).
Nom du fichier : **Nomdefamille_Prénom_L3_INF_S6_Algo.pdf**
Entête de la copie : **n° de la carte d'étudiant**, suivi du **Nom de famille de l'étudiant** suivis du prénom de l'étudiant. Ne pas mettre de titre ni de logo, les numéros de questions doivent être clairs.
En début de copie, noter toute remarque concernant le déroulement de l'examen que vous souhaitez porter à la connaissance du correcteur.

I. Débit maximal d'une route

~ 7 ***points***

Dans une période de télétravail intensif, les communications en visioconférence ont accru fortement les demandes en ressource réseau, en particulier en débit, pour pouvoir assurer le streaming des vidéos. On considère un réseau de communication constitué d'un ensemble de nœuds, les liens directionnels entre ces nœuds permettent de transmettre de l'information entre les nœuds, le débit crête d'un lien est la quantité maximale d'information que le lien est capable de transmettre par unité de temps. Sur une route entre un émetteur et un destinataire le débit est ainsi limité par le lien de plus faible débit crête (lien critique).

L'objectif du problème est de concevoir un algorithme construisant une route assurant un débit maximal entre un émetteur et un destinatire.



(a) Exemple de réseau avec débit limité

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	0	1	0	5	0	0	0
x_2	0	0	4	0	0	0	0
x_3	0	0	0	0	0	2	4
x_4	0	0	0	0	4	0	0
x_5	0	3	0	0	0	4	0
x_6	0	0	0	0	0	0	2
x_7	0	0	0	0	0	0	0

(b) la matrice des débits crête associée

Dans l'exemple ci-dessus la route $x_1 \to x_2 \to x_3 \to x_7$ admet un débit crête de 1 (limité par le lien (x_1, x_2)). La route $x_1 \to x_4 \to x_5 \to x_6 \to x_7$ admet un débit crête de 2 (limité par le lien (x_6, x_7)).

I.1. **Recherche d'une route (existence)**
Écrire une adaptation de l'algorithme de parcours en profondeur d'abord d'un graphe qui calcule, si elle existe, une route allant d'un nœud e (émetteur) à un nœud$d$ destinataire, e et d étant deux nœuds choisis préalablement. On précisera avec soin les structures de donnée auxiliaires utilisées. La route construite n'assure pas nécessairement le meilleur débit. [1]

I.2. **Exemple**
Illustrer par un dessin le déroulement de votre algorithme de parcours sur l'exemple ci-dessus.

On définit durant ce parcours et pour chaque nœud$x$ rencontré, $pere(x)$ le prédécesseur de x sur la route de e à x, $d(x)$ le débit de la route trouvée de e à x.

I.3. **Enrichissement de l'algorithme de parcours**
Compléter votre algorithme afin de construire les fonctions $pere$ et d pendant le parcours.

Parmi toutes les routes de s à x il en existe une, au moins, de débit maximal. Sur le graphe d'exemple, $x_1 \to x_4 \to x_5 \to x_2 \to x_3 \to x_7$ a un débit de 3 qui est maximal (limité par le lien (x_5, x_2)).

I.4. **Propriété des routes de débit maximal**
Montrer que si $e = x_{i_0} \to x_{i_1} \to \cdots \to x_{i_k} \to \cdots \to x_{i_l} = d$ est une route de débit maximal de e à d, $x_{i_k} \to \cdots \to x_{i_l} = d$ **n**'est **pas** nécessairement de débit maximal.

L'algorithme de Dijkstra étudié en cours permet de construire une arborescence de chemins de poids minimal dans un graphe pondéré (pondérations positives) ayant un sommet origine donné.

I.5. **Reconnaissance d'un problème classique**
Adapter l'algorithme de Dijkstra pour construire une route de débit maximal, on précisera la notion priorité utilisée.

I.6. **Exemple**
Illustrer par un dessin le déroulement de votre adaptation de l'algorithme de Dijkstra sur l'exemple ci-dessus.

I.7. **Condition sur le réseau**
L'exemple proposé ci-dessus est sans circuit, établir si cette hypothèse est nécessaire ou pas pour votre.

1. Rappel : pour explorer un graphe à partir d'un sommet x_0, on parcourt l'arbre des chemins issus de x_0 en marquant les sommets à leur première visite

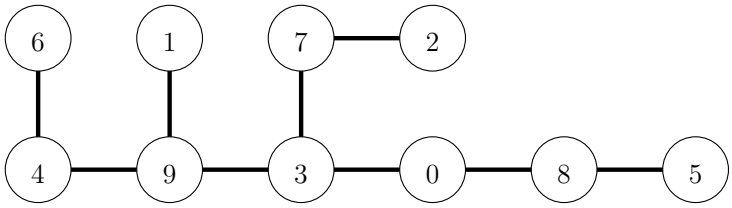
I.8. **Preuve**
Rédiger la preuve de votre algorithme, on précisera l'invariant associé à l'itération et on illustrera la démonstration par des dessins.

I.9. **Coût de l'algorithme**
Analyser la complexité de votre algorithme.

I.10. **Ensemble de toutes les routes** (question bonus)
On souhaite maintenant définir le débit maximal $d(x, y)$ pour tout couple (x, y) de nœuds du réseau. Proposer un algorithme qui construit la matrice $D = ((d(x, y))$ des débits maximaux et de complexité en $\mathcal{O}(n^3)$. Prouver votre algorithme et démontrer que sa complexité est bien en $\mathcal{O}(n^3)$.

II. Ranger un arbre dans une liste — ~ 7 *points*

On rappelle qu'un arbre est un graphe non orienté connexe et sans cycle. Dans ce problème on s'intéresse à la représentation d'un arbre comme une liste de sommets. Attention, les arbres envisagés ne sont pas forcément binaires ou d'arité fixée, il n'y a pas de racine. Par exemple, l'arbre ci-dessous est un arbre ayant $n = 10$ sommets, numérotés de 0 à 9



On considère l'algorithme suivant pour construire un arbre ayant n sommets, que l'on numérote de 0 à $n-1$, à partir d'une liste de $n-2$ sommets.

List_to_Tree (L)
Données : Une liste $L = [x_1, x_2, \cdots, x_{n-2}]$ de $n-2$ sommets, $0 \leqslant x_i \leqslant n-1$
Résultat : Un arbre ayant n sommets numérotés de 0 à $n-1$

Créer n sommets de 0 à $n-1$ et marquer les sommets par "non sélectionné"
for $i = 1$ **to** $n-2$ **do**
  $y \leftarrow$**Select_min** (L)
  `// y est le sommet non sélectionné de numéro le plus petit et qui n'est pas dans la sous-liste` $[x_i \cdots x_{n-2}]$ [a]
  Marquer y par "sélectionné"
  Créer une arête entre y et x_i
Créer une arête entre les deux derniers sommets "non sélectionnés"
return T

a. si les sommets déjà sélectionnés sont $1, 3, 7, 8$ et la sous-liste $[6, 5, 2, 0]$ alors le sommet à choisir sera le sommet numéro 4

Algorithme 1 : Algorithme de transformation d'une liste en arbre

II.1. **Un jeu de test**
À partir de votre numéro d'étudiant noté A, multipliez-le deux fois par π et prendre les 8 chiffres à gauche de la virgule. On obtient une liste de 8 chiffres qui servira d'entrée à l'algorithme **List_to_Tree**.
Par exemple, le numéro de l'étudiant Vifadacy est $A = 11710251$,

$$A \times \pi \times \pi = 1\underbrace{15575544}_{\text{8 chiffres}},8074610... \quad L = [1, 5, 5, 7, 5, 5, 4, 4]$$

Donner la liste L obtenue à partir de votre numéro d'étudiant (on détaillera les calculs comme ci-dessus).
Expliquer l'intérêt de faire une telle multiplication par π^2.

II.2. **Exemple de déroulement de l'algorithme**
Représenter l'exécution de l'algorithme **List_to_Tree** sur la liste L obtenue à la question précédente.

II.3. **Exemples particuliers**

Proposer quelques exemples particuliers permettant de comprendre la mécanique de cet algorithme, on justifiera le choix des exemples.

II.4. **Preuve de l'algorithme**

Faire la preuve complète de cet algorithme, on justifiera précisément l'invariant de boucle utilisé.

On souhaite maintenant écrire un algorithme qui transforme un arbre $\mathcal{A}$ dont les sommets sont numérotés de 0 à $n-1$ en une liste L telle que **List_to_Tree** $(L) = \mathcal{A}$

II.5. **Algorithme Tree_to_List**

II.5.a. Proposer un algorithme réalisant cette transformation

II.5.b. Vérifier que le résultat est correct sur l'arbre construit question 2

II.5.c. Illustrer sur l'exemple ci-dessus

II.5.d. Faire la preuve de votre algorithme

II.6. **Complexité** (question bonus)

Analyser la complexité de ces deux algorithmes. Peut-on utiliser des structures de données adaptées pour avoir une complexité en $\mathcal{O}(n)$ pour **List_to_Tree** ? et pour votre algorithme ?

III. Comparaison de différentes expressions d'un algorithme ~ 6 *points*

Durant toute la formation des étudiants en informatique, l'algorithme du tri rapide (Quicksort en anglais) est utilisé à de nombreuses reprises pour illustrer différents concepts, tri de données, tri en place, diviser pour régner, analyse en moyenne, récursivité,...

Cet algorithme est donc présenté dans de nombreux ouvrages d'algorithmique. L'objectif de ce problème est d'analyser les différentes présentations. Pour cela, il faut au préalable se donner des critères d'analyse de présentation d'algorithme, ces critères doivent recouvrir plusieurs points de vue que l'on peut avoir un un algorithme.

III.1. Établir une liste de critères permettant la comparaison de la présentation de cet algorithme.

III.2. Évaluer les cinq différentes présentations (ci-après) selon les critères proposés.

III.3. Faire une synthèse sur ces présentations.

Extrait 1 : Introduction à l'algorithmique 2nde édition, *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein,* Dunod (2002)

La procédure suivante implémente le tri rapide.

```
TRI-RAPIDE(A, p, r)
1  si p < r
2     alors q ← PARTITION(A, p, r)
3           TRI-RAPIDE(A, p, q − 1)
4           TRI-RAPIDE(A, q + 1, r)
```

Pour trier un tableau A entier, l'appel initial est TRI-RAPIDE$(A, 1, longueur[A])$.

a) Partitionner le tableau

Le point principal de l'algorithme est la procédure PARTITION, qui réarrange le sous-tableau $A[p\,..\,r]$ sur place.

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  pour j ← p à r − 1
4     faire si A[j] ⩽ x
5           alors i ← i + 1
6                 permuter A[i] ↔ A[j]
7  permuter A[i + 1] ↔ A[r]
8  retourner i + 1
```

Extrait 2 : Types de données et algorithmes, *Christine Froidevaux, Marie-Claude Gaudel, Michèle Soria*, McGraw-Hill (1990)

```
procedure tri-rapide(var t : array [1..n + 1] of integer; i, j : integer);
{on verra plus loin que t[n + 1] contient une sentinelle}
var k : integer;
begin
    if i < j then begin
        placer(t, i, j, k);
        {effectue la partition de t et le placement de t[i] en k}
        tri-rapide(t, i, k − 1);
        tri-rapide(t, k + 1, j)
    end
end tri-rapide;
```

L'appel de *tri-rapide*$(t, 1, n)$ provoque le tri de la liste complète.

```
procedure placer(var t : array [1..n + 1] of integer; i, j : integer;
                 var k : integer);
var l : integer;
begin
    l := i + 1; k := j;
    while l ≤ k do begin
        while t[k] > t[i] do k := k − 1; {t[k] ≤ t[i]}
        while t[l] ≤ t[i] do l := l + 1; {t[l] > t[i]}
        if l < k then begin t[l] ↔ t[k]; l := l + 1; k := k − 1 end
    end;
    t[i] ↔ t[k]
end placer;
```

Extrait 3 : Article Tri Rapide Wikipedia (25/05/2020) lien

```
partitionner(tableau T, entier premier, entier dernier, entier pivot)
    échanger T[pivot] et T[dernier]  // échange le pivot avec le dernier du tableau , le
pivot devient le dernier du tableau
    j := premier
    pour i de premier à dernier - 1 // la boucle se termine quand i = (dernier-1).
        si T[i] <= T[dernier] alors
            échanger T[i] et T[j]
            j := j + 1
    échanger T[dernier] et T[j]
    renvoyer j

tri_rapide(tableau T, entier premier, entier dernier)
        si premier < dernier alors
            pivot := choix_pivot(T, premier, dernier)
            pivot := partitionner(T, premier, dernier, pivot)
            tri_rapide(T, premier, pivot-1)
            tri_rapide(T, pivot+1, dernier)
```

Extrait 4 : Fundamentals of Algorithmics, *Gilles Brassard, Paul Bratley* Pearson (1995)

procedure *pivot*($T[i..j]$; **var** l)
{Permutes the elements in array $T[i..j]$ and returns a value l such that, at the end, $i \le l \le j$, $T[k] \le p$ for all $i \le k < l$, $T[l] = p$, and $T[k] > p$ for all $l < k \le j$, where p is the initial value of $T[i]$}
$p \leftarrow T[i]$
$k \leftarrow i$; $l \leftarrow j + 1$
repeat $k \leftarrow k + 1$ **until** $T[k] > p$ **or** $k \ge j$
repeat $l \leftarrow l - 1$ **until** $T[l] \le p$
while $k < l$ **do**
swap $T[k]$ and $T[l]$
repeat $k \leftarrow k + 1$ **until** $T[k] > p$
repeat $l \leftarrow l - 1$ **until** $T[l] \le p$
swap $T[i]$ and $T[l]$

Now here is the sorting algorithm. To sort the entire array T, simply call *quicksort*($T[1..n]$).

Extrait 5 : Algorithms 63 et 64, *C.A.R Hoare* Communication of the ACM n° 4, pages 321-322, 1961.

ALGORITHM 63
PARTITION
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  partition (A,M,N,I,J);  value M,N;
             array A;  integer M,N,I,J;
```

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure. Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

$M \leqq J < I \leqq N$ provided $M < N$
$A[R] \leqq X$ for $M \leqq R \leqq J$
$A[R] = X$ for $J < R < I$
$A[R] \geqq X$ for $I \leqq R \leqq N$

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

```
begin   real X;  integer F;
        F := random (M,N);  X := A[F];
        I := M;  J := N;
up:     for I := I step 1 until N do
                  if X < A [I] then go to down;
        I := N;
down:   for J := J  step −1 until M do
                  if A[J]<X then go to change;
        J := M;
change: if I < J then begin exchange (A[I], A[J]);
                             I := I + 1; J := J − 1;
                             go to up
                       end
else    if I < F then begin exchange (A[I], A[F]);
                             I := I + 1
                       end
else    if F < J then  begin exchange (A[F], A[J]);
                             J := J − 1
                       end;
end     partition
```

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  quicksort (A,M,N);  value M,N;
             array A;  integer M,N;
```

comment Quicksort is a very fast and convenient method of sorting an array in the random-access store of a computer. The entire contents of the store may be sorted, since no extra space is required. The average number of comparisons made is $2(M-N)\ln(N-M)$, and the average number of exchanges is one sixth this amount. Suitable refinements of this method will be desirable for its implementation on any actual computer;

```
begin     integer I,J;
          if M < N then begin partition (A,M,N,I,J);
                              quicksort (A,M,J);
                              quicksort (A, I, N)
                        end
end       quicksort
```