Nom :

Analyse d'algorithmes et programmation - 2e contôle continu

Enseignants: Christina Boura, Gaëtan Leurent

7 avril 2017

Durée du contrôle : 2h.

Les documents ainsi que l'accès aux programmes élaborés pendant les séances de TP sont autorisés.

Les quatre parties sont indépendantes.

Les questions d'algorithmique sont à faire sur papier ; les questions de programmation sont à faire sur machine, et la réponse sera envoyée par mail à `gaetan.leurent@inria.fr` *à la fin du contrôle.*

1 QCM : vrai ou faux ?

Exercice 1

Indiquer si chaque proposition est vraie, fausse, ou si c'est un problème ouvert.

	Vrai	Faux	On se sait pas

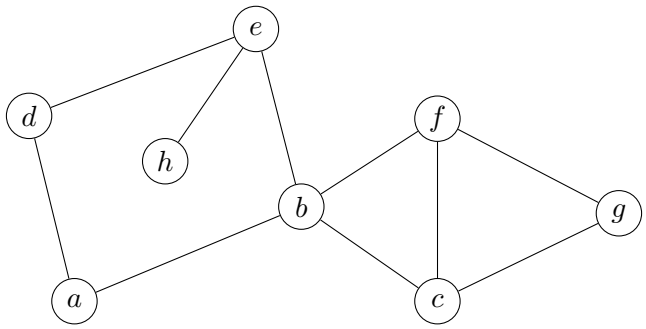
La fonction suivante calcule le n-ième nombre de Fibonacci :
`def fibo(n):`
`  if n==0:`
`  return 1`
`  else:`
`  return fibo(n-1)+fibo(n-2)`
	☐	☐	☐

La fonction suivante calcule la n-ième puissance de a :
`def power(a,n):`
`  if n==0:`
`  return 1`
`  else:`
`  return a*power(a,n-1)`
	☐	☐	☐

Il existe un algorithme pour multiplier deux matrices de taille n en temps $\mathcal{O}(n^3)$.	☐	☐	☐
Il existe un algorithme pour multiplier deux matrices de taille n en temps $\mathcal{O}(n^{2.1})$.	☐	☐	☐
Il existe un algorithme polynomial pour résoudre le problème 3-SAT (satisfaisabilité d'une formule sous forme normale conjonctive d'ordre 3).	☐	☐	☐
Il existe un algorithme polynomial pour résoudre le problème 2-SAT (satisfaisabilité d'une formule sous forme normale conjonctive d'ordre 2).	☐	☐	☐
$P \subseteq NP$	☐	☐	☐
$NP \subseteq P$	☐	☐	☐

2 Parcours de graphe

On considère le graphe suivant :

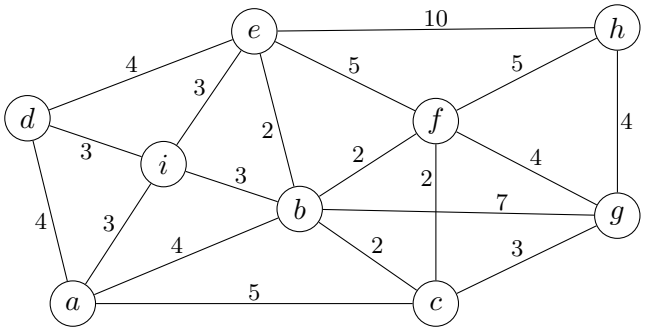


Exercice 2
Donner l'ordre de visite des sommets suivant un parcours en largeur commençant par le sommet "a".

Exercice 3
Donner l'ordre de visite des sommets suivant un parcours en profondeur commençant par le sommet "a".

3 Arbre couvrant minimal

On étudie un algorithme glouton pour construire un arbre couvrant minimal (différent de l'algorithme vu en cours). Cet algorithme parcours les arrêtes par ordre croissant de poids, et ajoute une arrête à l'arbre en cours de construction si et seulement si cela ne crée pas de cycle.
Exercice 4 (sur papier)
Appliquer cet algorithme sur le graphe suivant :



Marquer sur le graphe les arrêtes choisies, et donner l'ordre dans lequel elles sont choisies. Quel est le poids d'un arbre couvrant minimal ?

Pour détecter les cycles, on va maintenir une structure d'ensembles disjoints qui correspond aux composantes connexes de l'arbre en cours de construction. Initialement, chaque sommet est dans sa propre composante connexe, et quand une arrête est sélectionnée, les composantes des deux sommets sont fusionnées. Par exemple, on peut utiliser un attribut `.set` sur les noeuds, qui indique à quelle composante il appartient.

L'algorithme peut donc être décrit ainsi :

1: **fonction** ArbreCouvrant(G, w)
2: $A \leftarrow \varnothing$
3: **pour tout** sommet x **faire**
4: MakeSet(x)
5: $L \leftarrow \{\text{arrêtes } (u, v)\}$
6: Sort(L, w) ▷ Trier les arrêtes par poids
7: **pour** $0 \leq i < |L|$ **faire**
8: $(u, v) \leftarrow L[i]$
9: **si** FindSet(u) $\neq$ FindSet(v) **alors**
10: $A \leftarrow A \cup (u, v)$
11: FusionSet(u, v)
12: **retourner** A

On peut construire une structure d'ensembles disjoints tel que les opérations MakeSet, FindSet et FusionSet aient une complexité en $\mathcal{O}(\log n)$ pour une structure contenant n éléments.

Exercice 5 (sur papier)
Quel est alors la complexité de l'algorithme ?

Exercice 6 (sur machine)
Toto a implémenté l'algorithme en python, mais son programme ne marche pas.
Corriger le code de toto.
Note : une version informatique du code est disponible sur la page du cours.

```
def arbre_couvrant(G):
    for u in G.noeuds:
        u.set = u
    L = []
    for u in G.noeuds:
        for v,w in zip(u.out, u.weight):
            L.append(((u,v),w)) # Arrete (u,v) de poids w
    L.sort(key=(lambda x: x[1]))
    for (u,v),w in L:
        if u.set != v.set:
            Arbre.append((u,v))
            poids += w
            # Fusion des composantes: on remplace u.set par v.set
            for s in G.noeuds:
                if s.set == u.set:
                    s.set = v.set
    return Arbre,poids
```

Exercice 7 (sur papier)
Prouver par induction qu'à chaque itération de l'algorithme, il existe un arbre couvrant minimal qui contient A.
Indice : on se place au moment où une nouvelle arrête (u, v) est ajoutée dans A. En supposant qu'il existe un arbre couvrant minimal qui contient A, on construit un arbre couvrant minimal qui contient $A \cup (u, v)$.

4 Multiplication de polynômes

On veut calculer le produit de deux polynômes de degré n : $A = \sum_{i=0}^{n} a_i x^i$ et $B = \sum_{i=0}^{n} b_i x^i$.

$$C = A \times B = \sum_{i=0}^{2n} c_i x^i \qquad \text{avec : } c_i = \begin{cases} \sum_{j=0}^{i} a_j \cdot b_{i-j} & \text{si } i \leq n \\ \sum_{j=i-n}^{n} a_j \cdot b_{i-j} & \text{si } i \geq n \end{cases}$$

On représente un polynôme par la liste de ses coefficients $[a_0, a_1, ...a_n]$.

Exercice 8 (sur machine)
Implémenter une fonction `add(A,B)` *qui calcule la somme de deux polynômes.*

Exercice 9 (sur machine)
Implémenter une fonction `multiply_x(A,m)` *qui multiplie un polynôme par x^m.*

Pour calculer le produit, on utilise une décomposition en polynômes de degré $n/2$:

$$A = A_1 \cdot x^{n/2} + A_0 \qquad\qquad B = B_1 \cdot x^{n/2} + B_0$$

On peut alors calculer C avec

$$C = (A_1 \cdot x^{n/2} + A_0) \times (B_1 \cdot x^{n/2} + B_0)$$
$$C = \underbrace{(A_1 \times B_1)}_{C_2} \cdot x^n + \underbrace{(A_1 \times B_0 + A_0 \times B_1)}_{C_1} \cdot x^{n/2} + \underbrace{(A_0 \times B_0)}_{C_0}$$

On obtient ainsi l'algorithme diviser-pour-régner suivant :

fonction MULTIPLY(A, B)
  $\ell \leftarrow |A|$
  si $\ell = 1$ **alors**
    retourner $[A[0] \times B[0]]$
  sinon
    $A_0 \leftarrow A[0, \ldots, \ell/2 - 1]$
    $A_1 \leftarrow A[\ell/2, \ldots, \ell - 1]$
    $B_0 \leftarrow B[0, \ldots, \ell/2 - 1]$
    $B_1 \leftarrow B[\ell/2, \ldots, \ell - 1]$
    $C_0 \leftarrow$ MULTIPLY(A_0, B_0)
    $C_1 \leftarrow$ ADD(MULTIPLY(A_1, B_0), MULTIPLY(A_0, B_1))
    $C_2 \leftarrow$ MULTIPLY(A_1, B_1)
    retourner ADD$(C_0,$ MUTLIPLYX$(C_1, \ell/2),$ MULTIPLYX$(C_2, \ell))$

Exercice 10 (sur papier)
Quel est la complexité de cet algorithme ?

On peut améliorer cet algorithme en utilisant la relation :

$$C_1 = (A_0 + A_1)(B_0 + B_1) - C_2 - C_0$$

Exercice 11 (sur papier)
Décrire l'algorithme amélioré. Quel est sa complexité ?

Exercice 12 (sur machine)
Implémenter le nouvel algorithme en python.

Exercice 13 (sur machine)
Ajouter des tests pour gérer des polynômes de degré différents, et tels que ℓ ne soit pas une puissance de deux.