

| | |
|---|--|
|  <p>Master BioInformatique</p> | <p>ANNÉE : 2010/2011</p> <p>SESSION DE MARS 2011</p> <p>PARCOURS : Master 1</p> <p>UE TBS221 : Algorithmique 2ème partie</p> <p>Épreuve : Examen</p> <p>Date : Lundi 21 mars 2011</p> <p>Heure : 10 heures</p> <p>Durée : 2 heures</p> <p>Documents : autorisés</p> <p>Épreuve de M. Alain GRIFFAULT</p> |
|---|--|

Code d'anonymat :

Avertissement

- La plupart des questions sont indépendantes.
 - L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).

Exercice 1 (Automates de recherche de motifs (5 points))

Question 1.1 (2 points) Pour les mots sur l'alphabet $\Sigma = \{a, b, c\}$, dessinez l'automate de recherche du motif $abacabb$.

1. **What is the primary purpose of the study?** The study aims to evaluate the effectiveness of a new treatment for hypertension in a diverse population.

Question 1.2 (3 points) Pour les mots sur l'alphabet $\Sigma = \{a, b, c\}$, dessinez l'automate de recherche du motif $ab?cabb$ dans lequel le symbole $?$ remplace une et une seule des lettres de l'alphabet. L'automate cherche donc un des trois motifs $\{abacabb, abbcabb, abccabb\}$.

1. **What is the primary purpose of the study?** The study aims to evaluate the effectiveness of a new treatment for hypertension in a diverse population.

Exercice 2 (Graphes pondérés (5 points))

Question 2.1 (2 points) Soit $G(S, A, w)$ un graphe non orienté pondéré avec $w : A \rightarrow N$. Soit (u, v) une arête de poids minimal. Montrez que (u, v) appartient à un arbre couvrant de poids minimal.

Propriété 1 (admise) Soit $G(S, A, w)$ un graphe non orienté pondéré avec $w : A \rightarrow N$. L'algorithme de Dijkstra Dijkstra-PCC(G, s) vu en cours calcule :

- Pour chaque sommet u , la distance $d(s, u)$.
- Une arborescence des plus court chemins $\pi : S \rightarrow S \cup \{\text{nil}\}$

Propriété 2 (admise) Soit $G(S, A, w)$ un graphe non orienté pondéré avec $w : A \rightarrow Z$. L'algorithme de Dijkstra Dijkstra-PCC(G, s) vu en cours ne calcule pas toujours correctement

- Pour chaque sommet u , la distance $d(s, u)$.
- Une arborescence des plus courts chemins $\pi : S \rightarrow S \cup \{\text{nil}\}$ lorsque le poids de certaines arêtes est **négatif**.

Soit $G(S, A, w)$ un graphe non orienté pondéré avec $w : A \rightarrow Z$. Soit \min le plus petit poids porté par une arête. Soit $G'(S, A, w)$ un graphe non orienté pondéré avec la fonction $w'(a) = w(a) - \min$ pour toutes les arêtes a de A . Ainsi nous avons $w' : A \rightarrow N$.

Question 2.2 (3 points) Dijkstra-PCC(G', s) calcule pour G'

- Pour chaque sommet u , la distance $d'(s, u)$.
- Une arborescence des plus courts chemins $\pi' : S \rightarrow S \cup \{\text{nil}\}$

L'arborescence des plus courts chemins calculée pour (G', s) est-elle une arborescence des plus courts chemins pour (G, s) ? Justifiez votre réponse.

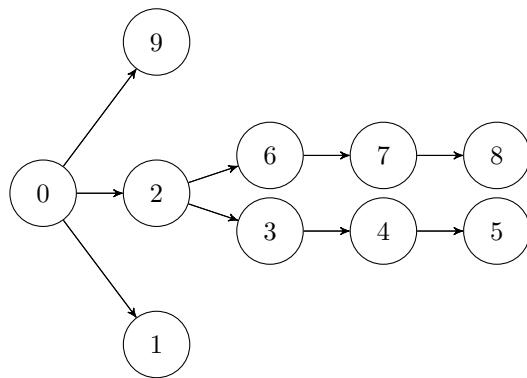


FIGURE 1 – Un arbre

Exercice 3 (Variantes parcours en profondeur (10 points))

Définition 1 (Diamètre) Soit $G(S, A)$ un graphe non orienté. Le diamètre du graphe G est la plus grande de toutes les distances des plus courts chemins du graphe. Formellement $\text{diametre}(G) = \max_{u,v \in S} \{d(u, v)\}$.

Propriété 3 (admise) Soit $T(S, A)$ un arbre non orienté. Soit (u, v) une arête de T . Notons T_u (resp. T_v) le sous arbre de $T - \{(u, v)\}$ de racine u (resp. v). Posons d_u le diamètre de T_u , d_v le diamètre de T_v , h_u la hauteur de T_u et h_v la hauteur de T_v . alors :

- $\text{diametre}(T) = \max(d_u, d_v, h_u + h_v + 1)$
- $\text{hauteur}(T) = \max(h_u, h_v + 1)$ si la racine de T est u
- $\text{hauteur}(T) = \max(h_v, h_u + 1)$ si la racine de T est v

Cette propriété permet le calcul du diamètre d'un arbre en utilisant deux attributs à chaque sommet lors d'un parcours en profondeur.

```

Visiter_PP(u) {
    // code classique de Visiter_PP
    u.couleur := gris;
    // attributs pour le calcul du diamètre
    u.diametre := 0;
    u.hauteur := 0;
    pour v dans u.adjacent faire {
        si (v.couleur = blanc)
        alors {
            Visiter_PP(v, t);
            u.diametre := max(u.diametre, v.diametre, u.hauteur + v.hauteur + 1);
            u.hauteur := max(u.hauteur, v.hauteur + 1);
        }
    }
    u.couleur := noir;
}
}

Diametre(G) {
    pour u dans S faire {
        u.couleur := blanc;
    }
    u := random(S(G)); // un sommet de G au hazard.
    Visiter_PP(u);
    retourner u.diametre;
}

```

Question 3.1 (2 points) Exécuter l'algorithme `Diametre(G)` sur l'arbre de la figure 1. Vous supposerez que `random(S(G))` retourne le sommet 0, et que les listes des adjacents sont triées dans l'ordre croissant des numéros.

| sommet | 0 | 1 | 0 | 2 | 3 | 4 | 5 | 4 | 3 | 2 | 6 | 7 | 8 | 7 | 6 | 2 | 0 | 9 | 0 |
|----------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| diamètre | | | | | | | | | | | | | | | | | | | |
| hauteur | | | | | | | | | | | | | | | | | | | |

Question 3.2 (2 points) L'algorithme `Diametre(G)` calcule t'il le diamètre d'un graphe non orienté dans le cas où G contient au moins un cycle. Vous justifierez votre réponse.

Définition 2 (Point d'articulation) Soit $G(S, A)$ un graphe non orienté connexe. Un point d'articulation de G est un sommet dont la suppression rend G non connexe.

Propriété 4 (admise) Soit $G(S, A)$ un graphe non orienté connexe. Soit s un sommet de G , et soit G_s l'arbre de racine s calculé par un parcours en profondeur $PP(G, s)$. s est un point d'articulation de G si et seulement si s possède au moins deux fils dans G_s .

Question 3.3 (2 points) Compléter l'algorithme afin que l'attribut `articulation` de la racine soit à vrai si la racine est un point d'articulation de G .

```

Visiter_PP(u) {
    // code classique de Visiter_PP
    u.couleur := gris;
    pour v dans u.adjacent faire {
        si (v.couleur = blanc)
        alors {
            v.pere := u;
            Visiter_PP(v);
        }
    }
    u.couleur := noir;
}
}

Articulation-Racine(G) {
    pour u dans S faire {
        u.couleur := blanc;
        u.pere := nil;
    }
    u := random(S(G)); // un sommet de G au hazard.
    Visiter_PP(u);
    // la racine u est-elle un point d'articulation
    // COMPLETEZ LE CODE

    retourner u.articulation;
}

```

Propriété 5 (admise) Soit $G(S, A)$ un graphe non orienté connexe. Soit G_s l'arbre de racine s calculé par un parcours en profondeur $PP(G, s)$. Soit u un sommet de G différent de s , u est un point d'articulation de G si et seulement si u possède un fils v dans G_s tel qu'il n'existe aucune arête retour partant de v ou d'un descendant de v et arrivant à un ancêtre propre de u .

Propriété 6 (admise) Soit $G(S, A)$ un graphe non orienté connexe. Soit G_s l'arbre de racine s calculé par un parcours en profondeur $PP(G, s)$. Soit pour chaque sommet $u \neq s$, la valeur $u.bas = \min(u.debut, \{w.debut\})$ où les w sont les sommets pour lesquels il existe une arête retour (v, w) dans G_s avec v descendant de u . u est un point d'articulation si et seulement si $u.bas \geq u.debut$.

Question 3.4 (4 points) Compléter l'algorithme afin que pour chaque sommet autre que la racine, l'attribut articulation soit à vrai si le sommet est un point d'articulation de G .

```

Visiter_PP(u) {
    // code classique de Visiter_PP
    u.couleur := gris;
    u.debut := date;
    date := date+1;
    // l'attribut bas est initialisé à la date de début de visite
    // et articulation à FAUX
    u.bas := u.debut;
    u.articulation = faux;
    pour v dans u.adjacent faire {
        si (v.couleur = blanc)
        alors {
            v.pere := u;
            Visiter_PP(v);
            // les attributs sont éventuellement mis à jour
            // COMPLETEZ LE CODE POUR U.BAS
            u.bas :=
        }
        // COMPLETEZ LE CODE POUR U.ARTICULATION
    }
    // pour une arête retour, l'attribut bas est éventuellement mis à jour
    // COMPLETEZ LE CODE
}

}
u.couleur := noir;
u.fin := date;
date := date+1;
}
}

Articulation(G) {
    pour u dans S faire {
        u.couleur := blanc;
        u.pere := nil;
        u.bas := 0
    }
    date := 0;
    u := random(S(G)); // un sommet de G au hasard.
    Visiter_PP(u);
}

```