Concours Commun INP (CCINP), France

Session 2024, filière MPI

Informatique

Durée : 4 heures

Ce que traite le sujet

- Partie I : algorithme Single Pass
- Partie II : langages réguliers
- Partie III : correspondance de Burge

Sujet reproduit sans retouche à partir de la page suivante.

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

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

ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE MPI

INFORMATIQUE

Durée : 4 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- *Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, bleu clair ou turquoise, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.*
- *Ne pas utiliser de correcteur.*
- *Écrire le mot FIN à la fin de votre composition.*

Les calculatrices sont interdites.

Le sujet est composé de trois parties, toutes indépendantes.

Partie I - Algorithme Single Pass

Cette partie comporte des questions nécessitant un code en **langage C**.

On cherche à classer $n = 5$ documents textuels $doc_i, i \in [\![1, n]\!]$ dans lesquels les termes T_1, T_2 et T_3 apparaissent un certain nombre de fois, ces occurrences étant décrites dans le tableau suivant :

	doc_1	doc_2	doc_3	doc_4	doc_5
T_1	1	2	0	1	1
T_2	3	1	1	3	0
T_3	3	0	0	1	1

Chaque document doc_i est donc représenté par un ensemble de $p = 3$ valeurs. On notera $d_i, i \in [\![1, n]\!]$ un vecteur de $\mathbb{N}^3$, de composantes $d_{ij}, j \in [\![1, p]\!]$, d_{ij} indiquant le nombre d'occurrences du terme T_j dans le document doc_i.
On cherche à voir si des textes traitent des mêmes thématiques, en faisant l'hypothèse que des textes sont sémantiquement proches si des termes communs apparaissent.

Q1. Recopier et remplir le tableau suivant en appliquant l'algorithme des k-moyennes avec $k = 2$ et d_2 et d_5 comme centres de classe initiaux. On utilisera la distance $\delta(d_i, d_j) = \sum_{\ell=1}^{p} |d_{i_\ell} - d_{j_\ell}|$. On notera de plus c_i les centres de classe et $\mathcal{A}_i$ les classes correspondantes.

Itération	c_1	c_2	$\mathcal{A}_1$	$\mathcal{A}_2$
1	d_2	d_5		
2				
3				

On souhaite maintenant traiter ce problème par une méthode dite "single pass" décrite dans l'**algorithme 1**.

Algorithme 1 - Algorithme Single Pass

Entrées : $(d_1 \cdots d_n)$ les vecteurs des documents, θ un seuil appartient à $\mathbb{R}$.
Sorties : $(\mathcal{A}_1 \cdots \mathcal{A}_j)$ les classes de centres $(c_1 \cdots c_j)$
début

```
c_1 = d_1 ;                                                    // Initialisation
A_1 = {d_1} ;
j = 1 ;
pour i de 2 à n faire
    pour k de 1 à j faire
        Étape (i) Calculer δ(d_i, c_k) ;
    si Étape (ii) δ(d_i, c_k) > θ ∀c_k alors
        j = j + 1 ;                                            // Création d'une nouvelle classe
        A_j = {d_i} ;
        c_j = d_i ;
    sinon
        // Indice du centre de classe le plus proche de d_i au sens de δ
        Étape (iii) ℓ = arg min_{1≤k<i} (δ(d_i, c_k)) ;
        A_ℓ = A_ℓ ∪ {d_i} ;                                    // Affectation de d_i à la classe ℓ
        c_ℓ = (1 / |A_ℓ|) Σ_{d_i ∈ A_ℓ} d_i ;                  // Recalcul de c_ℓ
```

Q2. Appliquer l'**algorithme 1** avec $\theta = 5.0$. Détailler les résultats des étapes de l'algorithme.

On propose d'implémenter cet algorithme en langage C. À cet effet, on définit un type structuré `vecteur` permettant d'encoder les centres de classe et les textes.

```
struct vecteur_s {
    double *v;          // pointeur vers les coordonnées
    int taille;         // taille du vecteur
    int num_classe;     // classe du vecteur
};
typedef struct vecteur_s vecteur;
```

Puisque le nombre de centres de classe varie au cours de l'algorithme, on utilise une liste chaînée de vecteurs pour représenter l'ensemble des centres de classe.

```
struct noeud_s {
    vecteur *c;
    struct noeud_s *suivant;
};
typedef struct noeud_s noeud;
```

Q3. Écrire une fonction de prototype `void ajoutVecteur(vecteur *vec, noeud **tete)` permettant d'ajouter un vecteur `vec` en tête de la liste des centres de classe pointée par `tete`. On prendra soin de vérifier que l'allocation mémoire s'est bien passée.

On suppose dans la suite disposer :

- de la variable `vecteur *documents[nb_documents]` qui contient l'ensemble des documents, où pour tout $i \in [\![0, \texttt{nb_documents} - 1]\!]$, `documents[i]` est égal à d_{i+1}.
- d'une fonction `void recalculCentre (vecteur *documents, noeud **tete, int l)` qui effectue le recalcul du centre c_ℓ et met à jour le nœud correspondant dans la liste chaînée des centres de classe.

Q4. Écrire une fonction de prototype `double delta(vecteur *di,vecteur *c)` qui calcule la distance entre le document `di` et le centre de classe `c` (étape **(i)** de l'**algorithme** 1). On suppose que d_i et c ont la même taille.

Q5. Écrire une fonction de prototype `bool distmax(double dists[],int j,double theta)` qui réalise l'étape **(ii)** de l'**algorithme** 1. Le tableau `dists` contient les distances de d_i à tous les c_k : pour tout $k \in [\![0, j-1]\!]$ l'élément `dists[k]` du tableau `dists` contient la distance de d_i à c_{k+1}. La fonction renvoie `true` si $\delta(d_i, c_k) > \theta \ \forall c_k$ et `false` sinon.

Q6. Écrire une fonction de prototype `int distmin(double dists[],int j)` qui réalise l'étape **(iii)** de l'**algorithme** 1. Le tableau `dists` contient les distances de d_i à tous les c_k comme dans la question précédente. La fonction renvoie l'entier l décrit dans l'algorithme.

Q7. À l'aide des questions précédentes et de la fonction `recalculCentre`, proposer une implémentation de l'**algorithme** 1 sous la forme d'une fonction de prototype
`noeud *algorithme1(vecteur *documents, int nb_documents, double theta)`.
Évaluer la complexité de l'**algorithme** 1.

Partie II - Langages réguliers

Soit $\Sigma = \{a, b, c\}$ un alphabet. Σ^* est l'ensemble des mots de longueur finie sur l'alphabet Σ. Pour $u \in \Sigma^*$, $|u| \in \mathbb{N}$ désigne la longueur du mot u. On note $u_i, i \in [\![1, |u|]\!]$ la i-ème lettre de u. Pour $k \in [\![1, |u|]\!]$, on note $u[1, k]$ le préfixe de u de longueur k, c'est-à-dire le mot $u_1 ... u_k$.
On définit sur Σ^* la relation symétrique $\mathcal{R}$ de la manière suivante : $u \mathcal{R} v$ s'il existe $w, z \in \Sigma^*$ tels que $u = wabz$ et $v = wbaz$. On note $\mathcal{R}^*$ la fermeture réflexive transitive de $\mathcal{R}$: $u \mathcal{R}^* v$ si $u = v$ ou s'il existe des mots $x_0 \cdots x_n$ tels que :

(i). $x_0 = u$,

(ii). pour $i \in [\![0, n-1]\!], i < min(|u|, |v|),\ x_i \mathcal{R} x_{i+1}$,

(iii). $x_n = v$.

Q8. Montrer que $\mathcal{R}^*$ est une relation d'équivalence.

Soit $u \in \Sigma^*$. La classe de u modulo $\mathcal{R}^*$ est l'ensemble des mots v vérifiant $v \mathcal{R}^* u$. Tous les mots de cette classe ont la même longueur. Ainsi, par exemple, {abcabb, abcbab, abcbba, bacabb, bacbab, bacbba} est une classe modulo $\mathcal{R}^*$.

Q9. Montrer que les classes modulo $\mathcal{R}^*$ sont des langages réguliers.

Q10. Donner les classes de mots de longueur 3.

On définit la relation $\preceq$ sur Σ telle que $a < b < c$ par : pour $u, v \in \Sigma^*$, $u \preceq v$ si l'une des deux conditions suivantes est réalisée :

(i). $u = v[1, |u|]$,

(ii). il existe $i > 1$ tel que $u[1, i-1] = v[1, i-1]$ et $u_i < v_i$.

Q11. Montrer que la relation $\preceq$ est réflexive et transitive. En déduire que $\preceq$ est un ordre total sur Σ^*.

Soit $\mathcal{U}$ une classe modulo $\mathcal{R}^*$. Le *représentant* de $\mathcal{U}$ est le plus petit élément de cette classe pour l'ordre $\preceq$. On note $\mathcal{L}$ l'ensemble des représentants des classes modulo $\mathcal{R}^*$.

Q12. Donner le représentant de la classe contenant le mot bacbab.

Q13. Donner une expression régulière du langage régulier $\mathcal{L}$.

Q14. Proposer un automate fini déterministe complet reconnaissant $\mathcal{L}$.

Partie III - Correspondance de Burge

La correspondance de Burge permet d'exprimer une bijection entre des graphes simples et des tableaux de Young semi-standards. Ces objets combinatoires ont de nombreuses applications, notamment dans l'étude de groupes symétriques et la géométrique algébrique. En théorie des graphes, ils permettent également de trouver, s'ils existent, les graphes simples dont les sommets sont contraints à avoir des degrés donnés.

Cette partie comporte des questions nécessitant un **code OCaml**. Pour ces questions, **les réponses ne feront pas appel aux fonctionnalités impératives du langage** (références, champs mutables, exceptions).

Notations
Dans toute la suite on notera :
- $[l] = [\![1, l]\!]$ l'ensemble des entiers naturels de 1 à l,
- $\left|E\right|$ le cardinal de l'ensemble E,
- $G = ([l], A)$ un *graphe simple*, c'est-à-dire un graphe non orienté à l sommets et m arêtes, sans boucles ni arêtes multiples,
- $d_i = \left|\{j \in [l], (i, j) \in A\}\right|$ le degré du sommet $i \in [l]$ dans le graphe $G = ([l], A)$.

De plus, les sommets de G seront ordonnés par degrés décroissants, de sorte que $d_1 \geq d_2 \geq \cdots \geq d_l \geq 0$.

Définition 1 (Suite de degrés d'un graphe)
Soit $G = ([l], A)$ un graphe simple. Si $d_i, i \in [l]$ est le degré du sommet i, avec $d_1 \geq d_2 \cdots \geq d_l \geq 0$, la *suite de degrés* de G est le l-uplet $(d_1, d_2 \cdots d_l)$.

III.1 - Partitions d'un entier

Définition 2 (Partition)
Une *partition* d'un entier $n \in \mathbb{N}^*$, notée $\lambda \vdash n$, est une suite décroissante $\lambda = (\lambda_1, \lambda_2, \cdots \lambda_l)$ de nombres entiers strictement positifs de somme n.

Par exemple, $n = 4$ a cinq partitions : (4), (3,1), (2,2), (2,1,1) et (1,1,1,1).

Une partition $\lambda = (\lambda_1 \cdots \lambda_l) \vdash n$ peut être vue comme une suite de degrés sous certaines conditions.

Q15. Montrer que si $\sum_{i=1}^{l} \lambda_i$ est impaire, alors la partition $\lambda \vdash n$ ne peut pas générer de graphe simple ayant la suite de degrés $(\lambda_1, \lambda_2 \cdots \lambda_l)$.

Dans la suite, on considérera que la somme des termes de la suite d'entiers $(\lambda_1, \lambda_2 \cdots \lambda_l)$ est paire.
Même avec cette contrainte, cette suite peut ne pas pouvoir générer de graphe simple.

Définition 3 (Suite graphique)
Une suite d'entiers $d_1 \geq d_2 \cdots \geq d_l \geq 0$ est dite *graphique* s'il existe un graphe simple dont la suite des degrés est $(d_1, d_2 \cdots d_l)$.

Notons qu'une suite graphique vérifie par définition $d_1 \geq d_2 \cdots \geq d_l$.

Pour déterminer si une suite d'entiers donnée est graphique, on peut utiliser l'algorithme d'Havel-Hakimi, basé sur le théorème suivant :

Théorème 1 (Havel-Hakimi)

(i). Pour tout $(d_1, ..., d_l) \in \mathbb{N}^l$, si $(d_1, \cdots, d_l)$ est graphique, alors $d_{d_1+1} > 0$ et toute permutation décroissante de $(d_2 - 1, d_3 - 1, \cdots, d_{d_1+1} - 1, d_{d_1+2}, \cdots, d_l)$ est graphique.

(ii). Réciproquement, pour tout $(d_2, \cdots, d_l) \in \mathbb{N}^{l-1}$, si $(d_2, \cdots, d_l)$ est graphique, alors pour $d_1 \in [\![d_2 + 1, l - 1]\!]$, la suite $(d_1, d_2 + 1, \cdots, d_{d_1+1} + 1, d_{d_1+2}, \cdots, d_l)$ est graphique.

Q16. Montrer que si $(d_2 \cdots d_l)$ est graphique, alors il existe un graphe simple $G = (S, A)$ à l sommets, de sommet 1 de degré $d_1 \in [\![d_2 + 1, l - 1]\!]$ tel que $(d_1, d_2 + 1 \cdots d_{d_1+1} + 1, d_{d_1+2}, d_{d_1+3}, \cdots d_l)$ soit la suite des degrés des sommets de G.

Q17. On suppose que $(d_1, d_2 \cdots, d_l)$ est graphique. Montrer qu'il existe $G = (S, A)$, $S = [l]$, le degré du sommet i étant d_i pour tout $i \in [l]$, tel que : $\forall j \in [\![2, d_1 + 1]\!]$, $(1, j) \in A$. Pour cela, on pourra raisonner par l'absurde et supposer que le sommet 1 est adjacent à un maximum de sommets dans $(2 \cdots d_1 + 1)$, mais pas à tous.

Q18. En déduire que si $d_{d_1+1} > d_{d_1+2}$, alors $(d_2 - 1, d_3 - 1 \cdots d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+3} \cdots d_l)$ est graphique.

Q19. Déterminer si les suites (4,3,3,3,3) et (6,4,4,2,2,1,1) sont graphiques.

Q20. Écrire une fonction de signature `compare_entiers : int -> int -> int` qui compare deux entiers et telle que l'appel à `compare_entiers m n` renvoie 1 si `m< n`, -1 si `m> n` et 0 sinon.

Q21. Écrire une fonction de signature `decr_n : n -> int list -> int list option` telle que pour tout $n \geq 0$ l'appel `decr_n n l` retourne `Some l'`, avec `l'` la liste `l` dont les n premiers éléments sont décrémentés, s'ils étaient tous supérieurs à 1 et `None` sinon.

Q22. Écrire une fonction récursive de signature `havel_hakimi : int list -> bool` qui retourne `true` si la liste passée en entrée, triée par ordre décroissant, est graphique et `false` sinon. À chaque étape utilisant le **théorème 1**, il sera nécessaire de réordonner par ordre décroissant la liste `list` construite, ce qui pourra être fait à l'aide de l'appel à `List.sort compare_entiers list`.

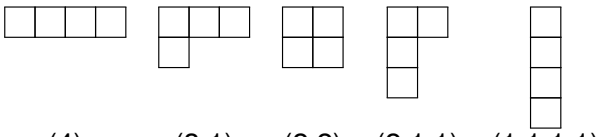
Q23. Donner deux graphes simples à $l = 5$ sommets ayant une même suite de degrés $(3, 2, 2, 2, 1)$.

Ainsi, il n'existe pas de correspondance univoque entre suite de degrés et graphe simple.

III.2 - Diagramme et tableau de Young

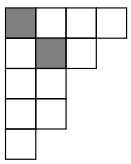
Définition 4 (Diagramme de Young d'une partition)
Le diagramme de Young de forme λ, noté $Y(\lambda)$, d'une partition $\lambda = (\lambda_1, \lambda_2, \cdots \lambda_l) \vdash n$ est un tableau de cases constitué de l lignes alignées à gauche, chaque ligne $i \in [l]$ ayant λ_i cases.
Par convention, la ligne associée à λ_1 est la première ligne du tableau.

Par exemple, les diagrammes de Young des partitions de l'entier 4 sont



Définition 5 (Diagonale d'un diagramme de Young)
Soit $Y(\lambda)$ un diagramme de Young. La diagonale de $Y(\lambda)$ est définie par les r cases $(i, i), i \in [r]$.

Dans le tableau suivant, $r = 2$ et les cases de la diagonale sont grisées.



Définition 6 (Partition conjuguée)
Soit $Y(\lambda)$ un diagramme de Young associé à $\lambda \vdash n$. La partition conjuguée de λ, notée $\lambda^* = (\lambda_1^*, \cdots, \lambda_k^*)$, est la partition obtenue en énumérant le nombre de cases de $Y(\lambda)$ par colonne, en partant de la colonne de gauche.

On remarque immédiatement que pour tout j, $\lambda^*_j = \left|\{i, \lambda_i \geq j\}\right|$.

Q24. Soit $\lambda = (5,4,1,1,1,1) \vdash 13$. Donner λ^*.

Définition 7 (Représentation de Frobenius d'un diagramme de Young)
Soient $(\lambda_1, \lambda_2, \cdots \lambda_l) \vdash n$ une partition, $Y(\lambda)$ son diagramme de Young et r la taille de sa diagonale. Pour $i \in [r]$, soient $\alpha_i = \lambda_i - i$ le nombre de cases à droite de la case (i,i) dans la i-ème ligne de $Y(\lambda)$ et $\beta_i = \lambda^*_i - i$ le nombre de cases en-dessous de la case (i,i) dans la i-ème colonne de $Y(\lambda)$. Alors $\alpha_1 > \alpha_2 \geq \cdots \alpha_r \geq 0$, $\beta_1 > \beta_2 \geq \cdots \beta_r \geq 0$ et la notation de Frobenius de λ est donnée par $\lambda = \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots \alpha_r \\ \beta_1 & \beta_2 & \cdots \beta_r \end{pmatrix}$.

Q25. Soit $(4,3,2,2,1) \vdash 12$. Donner la représentation de Frobenius de $Y(\lambda)$.

Définition 8 (Tableau de Young)
Soit $\lambda \vdash n$ une partition de $n > 0$. Un λ-tableau de Young est un tableau obtenu en remplissant les cases d'un diagramme Y(λ) par des entiers dans $[n]$.

Par exemple, pour $\lambda = (2,1) \vdash 3$, les tableaux suivants sont des λ-tableaux

$$\begin{array}{|c|c|}\hline 1&2\\\hline 3\\\cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 2&1\\\hline 3\\\cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 3&1\\\hline 2\\\cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 2&3\\\hline 1\\\cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 3&2\\\hline 1\\\cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1&1\\\hline 2\\\cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1&3\\\hline 1\\\cline{1-1}\end{array}$$

Dans la suite, on notera $T(i,j)$ l'entier en position (i,j) dans le tableau T.

Définition 9 (Tableau de Young (semi)standard)
Un tableau de Young est dit *semi standard* si les éléments de chaque ligne (respectivement colonne) forment une suite croissante de gauche à droite (respectivement strictement croissante de haut en bas). Il est dit *standard* s'il est semi-standard et si les entiers de 1 à n n'apparaissent qu'une et une seule fois.

Dans les exemples précédents, les premier, troisième et septième tableaux sont semi standards. Les premier et troisième tableaux sont standards.

III.3 - Insertion de Schensted

Pour construire un tableau de Young semi-standard à partir d'une suite d'entiers $d_1, \cdots d_l$, on utilise une méthode d'insertion proposée par Schensted.
On cherche à insérer dans un tableau de Young semi standard T un entier k, de sorte à ce que le tableau créé (contenant une case de plus) soit toujours un tableau de Young semi standard. On note cette opération d'insertion $T \leftarrow k$.
L'entier k est inséré dans T en le comparant aux valeurs de la première ligne et en déplaçant le premier entier plus grand que k en lisant la ligne de gauche à droite. S'il n'y a pas de plus grand élément, k est ajouté à la fin de la ligne. Si un élément est déplacé, il est inséré dans la deuxième ligne et les lignes suivantes du tableau sont traitées de la même manière. L'**algorithme 2** décrit l'insertion de k dans la i-ème ligne de T.

Soit T_0 le tableau de Young semi standard $\begin{array}{|c|c|c|c|c|}\hline 1&1&2&2&4\\\hline 2&3&3\\\cline{1-3}3\\\cline{1-1}4\\\cline{1-1}\end{array}$

Q26. Insérer la valeur 5 dans T_0 et donner le tableau de Young semi standard résultant. Même question pour l'insertion de la valeur 1 dans T_0.

Q27. Énoncer une précondition de l'**algorithme 2** permettant de prouver sa correction, c'est-à-dire qu'il retourne bien un tableau de Young semi standard contenant k.

Pour coder les tableaux de Young, on choisit d'utiliser un type OCaml `type tableau = int list list` et on encode la structure par liste de lignes.

Algorithme 2 - Algorithme d'insertion d'un entier k dans la ligne i d'un tableau de Young semi standard

Insertion(T,k,i)
Entrées : un tableau de Young semi standard T, un entier k à insérer, i la ligne traitée
Sorties : un tableau de Young semi standard contenant k
début
  si *(la ligne i est vide) OU (k plus grand que l'élément le plus à droite de la ligne i)* **alors**
    Ajouter k en bout de la ligne i de T
  sinon
    Soit j le plus petit indice tel que $k < T(i, j)$
    $p = T(i, j)$
    $T(i, j) = k$
    Insertion(T,p,$i + 1$)

Q28. Écrire une fonction récursive de signature `insereligne :int -> int list -> int*int list` qui, à partir d'un entier k et d'une ligne i d'un tableau de Young semi standard, retourne un couple (m, r) où
- $m = 0$ et r est la ligne i où k a été ajouté en queue, si k est plus grand que tous les éléments de la ligne i,
- m, qui est dans la ligne i et r est la ligne i où m a été remplacé par k sinon.

Q29. En déduire une fonction récursive de signature `insertion : int -> tableau -> tableau` réalisant l'**algorithme 2**.

Q30. Écrire une fonction récursive de signature `construit_tableau :int list -> tableau` qui construit un tableau semi standard à partir d'une suite d'entiers.

L'**algorithme 2** permet de définir une position (s, t), coordonnées de la case où k a été ajouté à T.

III.4 - Correspondance de Burge

Définition 10 (Tableau de Burge)
Soit $G = ([l], A)$ un graphe simple, $|A| = m$. Le *tableau de Burge* associé à G est défini par

$$\mathcal{B}_G = \begin{pmatrix} u_1 & u_2 & \cdots u_m \\ v_1 & v_2 & \cdots v_m \end{pmatrix}$$

où pour tout $i \in [m]$ (u_i, v_i) est une arête de G, avec $u_i > v_i$, le tableau étant formé de sorte que pour $i \in [m-1]$, $u_i \leq u_{i+1}$ et si $u_i = u_{i+1}$ alors $v_i > v_{i+1}$.

Soit le graphe G de la **figure 1**.

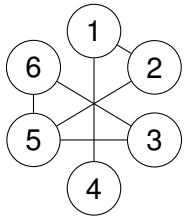


Figure 1 - Graphe exemple

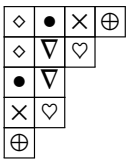
Q31. Donner le tableau $\mathcal{B}_G$ correspondant au graphe de la **figure 1**.

On utilise alors $\mathcal{B}_G$ pour associer à G un diagramme de Young $Y(\lambda)$ représenté par une forme de Frobenius particulière, notée $\mathcal{F}_Y = \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots \alpha_r \\ \alpha_1 + 1 & \alpha_2 + 1 & \cdots \alpha_r + 1 \end{pmatrix}$, où r est le nombre de cases de la diagonale principale du diagramme de Young $Y(\lambda)$ associé.

Q32. Montrer que, dans ce cas, pour tout $i \in [r]$ $\lambda_i^* = \lambda_i + 1$.

Ainsi, $Y(\lambda)$ est divisé en deux parties symétriques. La partie inférieure est constituée de toutes les cases qui se trouvent strictement sous la diagonale et la partie supérieure est constituée du reste. Chaque position dans la partie supérieure (inférieure) du diagramme de Young correspond à une position unique, appelée *position opposée*, dans la partie inférieure (supérieure) de $Y(\lambda)$. Par exemple, dans le diagramme de Young suivant, ayant comme représentation de Frobenius $\mathcal{F}_Y$, les positions opposées sont codées par le même symbole.

$$\mathcal{F}_Y = \begin{pmatrix} 3 & 1 \\ 4 & 2 \end{pmatrix}$$



Q33. Soit (s,t) une case. Donner la position opposée en fonction de s et t.

L'**algorithme 3** utilise alors la représentation $\mathcal{B}_G$ d'un graphe G pour construire un tableau de Young semi standard dont le diagramme correspondant $Y(\lambda)$ admet une représentation de Frobenius du type $\mathcal{F}_Y$. Dans cet algorithme, $T \leftarrow v_i$ insère la valeur v_i dans le tableau de Young semi standard T.

Algorithme 3 - Algorithme de Burge

Entrées : $\mathcal{B}_G$ de taille $m \times 2$

Sorties : tableau de Young semi standard T dont la représentation de Frobenius du diagramme correspondant est du type $\mathcal{F}_Y$

début

```
T = tableau vide ;                                   // Initialisation
pour i de 1 à m faire
    (s,t) = T ← v_i
    Placer u_i dans la position opposée à (s,t)
```

Cet algorithme produit, à partir du graphe de la **figure 1**, le tableau :

1	1	2	3
2	3	5	
4	5		
5	6		
6			

Q34. Donner les tableaux de Young semi standard intermédiaires produits à chaque itération.

Q35. À quoi correspond le nombre d'apparitions de chaque entier contenu dans les cases du tableau de Young résultat de l'**algorithme 3** ?

Notons pour terminer qu'il est à l'inverse possible de produire un tableau bidimensionnel $\mathcal{B}_G$ (et donc un graphe G) à partir d'un tableau de Young semi standard $Y(\lambda)$ de type $\mathcal{F}_Y$.

FIN

IMPRIMERIE NATIONALE – 24 1124 – D'après documents fournis

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
- Langage OCaml : expressions, types, filtrage, récursivité

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/ccinp/mpi