Concours e3a-Polytech, France

Session 2018, filière MP

Informatique

Durée : 3 heures

Ce que traite le sujet

- Exercice 1 (Python) : terminaison et problème de l'arrêt
- Exercice 2 (Caml) : valuation, résidu et pgcd
- Exercices 3 et 4 (Caml, SQL) : chemins dans un graphe et base de zoos
- Exercice 5 (Caml) : formes normales conjonctives positives

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/e3a/mp/2018/info

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

CONCOURS ARTS ET MÉTIERS ParisTech - ESTP - POLYTECH

Épreuve d'Informatique MP

Durée 3 h

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

L'usage de calculatrices est interdit.

AVERTISSEMENT

L'épreuve est composée de 5 exercices, totalement indépendants. Le langage à utiliser pour un exercice est indiqué à côté du numéro de l'exercice.

Un candidat pourra toujours admettre le résultat des questions qu'il n'a pas faites pour faire les questions suivantes.

La **présentation**, la lisibilité, l'orthographe, la qualité de la **rédaction, la clarté et la précision** des raisonnements entreront pour une **part importante** dans **l'appréciation des copies**. En particulier, les résultats non justifiés ne seront pas pris en compte. Les candidats sont invités à encadrer les résultats de leurs calculs.

Tournez la page S.V.P.

Il est interdit aux candidats de signer leur composition ou d'y mettre un signe quelconque pouvant indiquer sa provenance.

Exercice 1 (Python)

On dit qu'une fonction *termine* si elle renvoie une valeur ou si elle lève une exception (par exemple `ZeroDivisionError`). Une fonction peut terminer ou continuer à calculer à l'infini. Ainsi, la fonction `fun1` (voir ci-dessous), pour un entier `n`, termine pour `n` inférieur ou égal à `10` (elle renvoie `None`) et ne termine pas pour `n` strictement plus grand que `10`. En outre, cette fonction termine pour une chaîne de caractères `n` (en levant l'exception `TypeError`).

1. Pour quelles valeurs de `n` dans $\mathbb{Z}$ la fonction `fun2` termine-t-elle ? Même question pour `fun3`.

```
def fun1 (n):
    while n != 10 :
        n = n + 1
```

```
def fun2 (n):
    if n % 2 == 0:
        return 0
    else :
        while True:
            n = n + 1
```

```
def fun3 (n):
    S = 0
    while n != 0 :
        S = S + n
        n = n - 2
    return S
```

2. Détailler l'exécution de `fun3(10)`.

3. Écrire en Python une fonction `ForEver(n)` qui ne termine jamais.

Le module `dis` permet, à partir du nom d'une fonction, de trouver quel est le bytecode Python qu'elle exécute, et d'analyser ce bytecode. On se demande quelles propriétés sur la fonction peuvent être déduites de cette analyse. Plus précisément, nous souhaitons répondre au problème suivant.

Problématique

Est-il possible d'écrire en Python une fonction `arret`, qui termine toujours, et telle que `arret(f,x)` renvoie `True` si le calcul de `f(x)` termine et `False` sinon ?

4. Dans cette question, nous supposons qu'une telle fonction `arret` existe.
   (a) Écrire en Python une fonction `strange(f,x)` qui termine si et seulement si le calcul de `f(x)` ne termine pas.
   (b) Écrire en Python une fonction `paradox(f)` qui termine si et seulement si le calcul de `f(f)` ne termine pas.

5. Le calcul de `paradox(paradox)` termine-t-il ? Qu'en déduire quant à l'existence de `arret` ?

Exercice 2 (Caml)

Tout entier naturel non nul n s'écrit de manière unique $n = 2^v k$ avec v un entier naturel (qui peut valoir zéro) et k un entier naturel impair. Dans la suite de cet exercice, nous appelons valuation de n l'entier v et résidu de n l'entier k. Par convention, le résidu de zéro est zéro.

1. Expliquer succintement comment, à partir de l'écriture en base deux de $n \in \mathbb{N}^\star$, on peut lire la valuation et le résidu de n.
2. L'entier 192 s'écrit en base deux 11000000. Donner sa valuation et son résidu.
3. Écrire en Caml une fonction `residu : int -> int` qui prend en argument un entier positif **ou nul** et renvoie son résidu.

Pour calculer le pgcd (plus grand commun diviseur) de deux entiers, nous pouvons utiliser l'algorithme des soustractions successives décrit ci-après.

```
1  Entrées : deux entiers naturels non nuls a et b
2  Tant que b est non nul :
3        Remplacer b par |a − b|.
4        Remplacer a par le minimum de a et de l'ancienne valeur de b.
5  Fin du "Tant que".
6  Renvoyer a
```

4. Écrire en Caml une fonction `pgcd1 : int -> int -> int` qui calcule le pgcd de deux entiers naturels non nuls en utilisant cet algorithme et pas un autre. Cette fonction ne doit pas être récursive ni faire appel à une ou des fonctions auxiliaires récursives.
5. Écrire en Caml une fonction **récursive** `pgcd2 : int -> int -> int` qui calcule le pgcd de deux entiers naturels non nuls en utilisant la méthode des soustractions successives.
6. Estimer (en justifiant) la complexité de cet algorithme en fonction de $n = \max(a, b)$.

La méthode chinoise, mentionnée dans *Les Neuf Chapitres sur l'art mathématique* écrit aux débuts de la dynastie Han, consiste à remplacer la ligne 3 par la ligne suivante :

```
3        Remplacer b par le résidu de |a − b|.
```

On admet que cette méthode permet de calculer le pgcd de a et b si a ou b est **impair**. Ce résultat peut notamment être utilisé pour répondre à la question suivante.

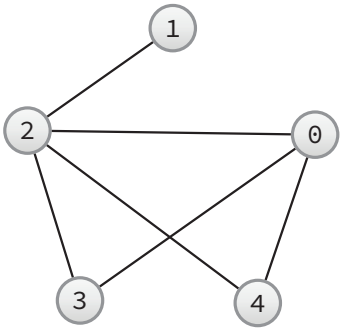
Tournez la page S.V.P.

7. Que calcule la méthode chinoise lorsque a et b sont **pairs** ? Le démontrer laconiquement.
   Indication : On peut distinguer le cas $a = b$ du cas $a \neq b$.

8. Estimer (en justifiant) la complexité de cette méthode en fonction de $n = \max(a, b)$, dans le cas où a et b sont tous les deux impairs.

9. En déduire une fonction `pgcd_chinois : int -> int -> int` qui calcule le pgcd de deux entiers (pairs ou impairs) avec une complexité du même ordre de grandeur que la complexité calculée la question précédente. Justifier la complexité de `pgcd_chinois`.

Exercice 3 (Caml)



1. Écrire la matrice d'adjacence du graphe ci-dessus.

2. Écrire en Caml une fonction `chemin : int vect vect -> int list -> bool` qui prend en entrée la matrice d'adjacence d'un graphe et un chemin (une liste de sommets du graphe) et qui vérifie si ce chemin est possible dans le graphe. Par exemple, sur le graphe ci-dessus, avec le chemin `[2;1;0;4]` la fonction `chemin` doit renvoyer `false` car les sommets `1` et `0` ne sont pas connectés. Avec le chemin `[1;2;3]` la fonction `chemin` doit renvoyer `true` car les sommets `1` et `2` sont connectés, ainsi que les sommets `2` et `3`.

Exercice 4 (SQL)

Nous nous intéressons à une base de données des zoos qui contient deux tables. La première table, `zoos`, a quatre colonnes dont `id`, un identifiant unique pour chaque zoo. Quelques lignes sont données ci-après.

id	nom	pays	continent
FR42	Zoo de La Flèche	France	Europe
RU12	Parc zoologique de Novossibirsk	Russie	Asie
RU5	Parc zoologique de Saint-Pétersbourg	Russie	Europe
⋮	⋮	⋮	⋮

La seconde table, `animaux`, a 6 colonnes, notamment un identifiant unique pour chaque animal (`id`) et l'identifiant du zoo qui héberge l'animal (`zoo`).

id	nom	espece	sexe	naissance	zoo
ke860	Kaiko	Chameau	F	2013	FR42
ic431	Jeffrey	Python royal	M	2016	RU12
gz599	Antaeus	Annaconda vert	M	2016	RU12
⋮	⋮	⋮	⋮	⋮	⋮

Les questions suivantes demandent d'écrire des requêtes SQL. À chaque fois quelques lignes de la table attendue sont données en exemple .

1. Écrire une requête SQL renvoyant la table des chamelles (chameaux femelles).

id	nom	naissance	zoo
ke860	Kaiko	2013	FR42
md375	Aimy	2012	CG01
⋮	⋮	⋮	⋮

2. Écrire une requête SQL renvoyant la table des bonobos mâles vivant en Asie.

id	nom	naissance	zoo
yv919	Finn	2008	CN33
qv139	Proteus	2013	KR08
⋮	⋮	⋮	⋮

3. Écrire une requête renvoyant la liste des pays ayant des zoos sur plusieurs continents.

pays
Russie
⋮

Exercice 5 (Caml)

Une *formule positive* est une formule propositionnelle n'utilisant comme connecteurs logiques que "ou" et "et". Ces connecteurs sont notés, respectivement, $\vee$ et $\wedge$. On représente les formules positives en Caml par le type suivant :

Tournez la page S.V.P.

```
type fp = OU of fp * fp | ET of fp * fp | VAR of string ;;
```

1. Considérons la formule "$X \vee (Y \wedge Z)$" et notons-la φ_0.
   (a) Dessiner l'arbre correspondant à φ_0.
   (b) Écrire φ_0 en Caml en utilisant le type `fp`.

On définit par récurrence les *disjonctions de variables propositionnelles* (DVP) ainsi :

- Si X est une variable propositionnelle, alors X est une DVP.
- Si φ et ψ sont des DVP alors $\varphi \vee \psi$ est aussi une DVP.

2. Écrire en Caml une fonction `dvp : fp -> bool` prenant en argument une formule positive `f` et renvoyant `true` si et seulement si `f` est une DVP.

On appelle *forme normale conjonctive positive* (FNCP) une conjonction de DVP. Ainsi $(X \vee U) \wedge (X \vee Y \vee Z) \wedge T$ est une forme normale conjonctive positive, mais pas $(X \wedge U) \vee (X \wedge (Y \vee Z))$. Plus précisément, on définit les FNCP par récurrence comme suit :

- Si φ est une DVP alors φ est une FNCP.
- Si φ et ψ sont des FNCP, alors $\varphi \wedge \psi$ aussi.

3. Écrire en Caml une fonction `fncp` de type `fp -> bool` prenant en argument une formule positive `f` et renvoyant `true` si et seulement si `f` est une FNCP.

On définit la fonction `norm` qui, étant donné une formule positive `f`, renvoie une FNCP logiquement équivalente à `f`.

```
let rec norm f = match f with                                          0
        VAR _ -> f                                                     1
        | ET(a,b) -> ET (norm a, norm b)                               2
        | OU(ET(a,b),c) -> ET( norm (OU(a,c)) , norm (OU(b,c)) )       3
        | OU(c,ET(a,b)) -> ET( norm (OU(a,c)) , norm (OU(b,c)) )       4
        | OU (a,b) -> let c = OU( norm a, norm b) in                   5
              if f=c                                                   6
              then f                                                   7
              else norm c;;                                            8
```

4. Expliquer brièvement pourquoi si `f` est une FNCP alors `norm f` termine et renvoie `f`.

5. Montrer que si `norm f` termine, alors elle renvoie une FNCP.

6. Exhiber, sans démonstration, une fonction simple λ de l'ensemble des formules dans $\mathbb{N}$ telle que pour toutes formules a, b, c et d :

   - $\lambda(a \vee b) = \lambda(a \wedge b) > \max(\lambda(a), \lambda(b))$,
   - $\lambda((a \wedge b) \vee c) = \lambda(c \vee (a \wedge b)) > \max(\lambda(a \vee c), \lambda(b \vee c))$,
   - Si $\lambda(a) \geqslant \lambda(c)$ et $\lambda(b) \geqslant \lambda(d)$ alors $\lambda(a \vee b) \geqslant \lambda(c \vee d)$.

On introduit la fonction μ de l'ensemble des formules dans $\mathbb{N}$. $\mu(\varphi)$ est la profondeur minimale d'un nœud ET dans la formule φ. Plus précisément, μ est définie comme suit :

- $\mu(X) = 0$ si X est une variable propositionnelle,
- $\mu(\varphi \wedge \psi) = 1$ pour toutes formules φ et ψ,
- $\mu(\varphi \vee \psi) = \min\big(\mu(\varphi), \mu(\psi)\big)$ pour toutes formules φ et ψ.

7. On considère une formule f telle que :

   - f correspond au cas de filtrage (*pattern matching* en anglais) de la ligne 5 mais ne correspond à aucun des cas des lignes 1 à 4,
   - Le calcul de c à la ligne 5 termine.

   (a) Montrer que si $f \neq c$ alors $\mu(f) > \mu(c)$.
   (b) Montrer que $\lambda(f) \geqslant \lambda(c)$.

8. Démontrer soigneusement que `norm` termine sur toutes les formules positives.

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP 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

- Spécifier, tester, prouver : terminaison et correction
- Bases de Python : expressions, types, instructions, fonctions
- Langage OCaml : expressions, types, filtrage, récursivité
- Complexité des algorithmes itératifs et récursifs
- Architecture de l'ordinateur et représentation des nombres
- Graphes : représentations et parcours
- Logique, ensembles et applications

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/e3a/mp