Épreuve d'informatique de l'X - 2012 - PT/PSI

Partie I. Autour des ensembles

Question 1. Il s'agit de compter le nombre de `vrai` présents dans le tableau. On parcourt le tableau dans la boucle des lignes 3 à 5 et on incrémente le compteur `cmpt` si i appartient à l'ensemble représenté par le tableau t (ligne 5).

```
1 def cardinal(t):
2     cmpt = 0;
3     for i in range(taille(t)):
4         if t[i]:
5             cmpt +=1
6     return(cmpt)
```

Question 2. Si i est plus grand que la taille du tableau t alors i n'appartient pas à l'ensemble représenté par le tableau t (ligne 5). Sinon, il suffit de retourner la valeur $t[i]$ (ligne 3).

```
1 def appartient(i,t):
2     if i < taille(t):
3         return(t[i])
4     else:
5         return(False)
```

Question 3 : On commence par créer un tableau de la même longueur que le tableau t_1 (ligne 3). Puis, par la boucle des lignes 4-5, on « remplit » le tableau résultat `res`. On met `true` dans `res`$[i]$ si i appartient à S_1 mais n'appartient pas à S_2. Dans le cas contraire, on met `false`.

```
1 def diff(t1,t2):
2     n1 = taille(t1)
3     res = allouer(n1,False)
4     for i in range(n1):
5         res[i] = t1[i] and not(appartient(i,t2))
6     return(res)
```

Question 4 : On commence récupérer dans la variable n la plus grande des deux longueurs de tableau (lignes 2 à 5). Ensuite, on crée un tableau de la même longueur que le plus grand des tableaux (ligne 6). Dans la boucle des lignes 7 à 8, on met `true` dans `res`$[i]$ si i appartient à S_1 ou à S_2. Dans le cas contraire, on met `false`. On retourne le résultat à la ligne 8.

```
1 def union(t1,t2):
2     if taille(t1) > taille(t2):
3         n = taille(t1)
4     else:
5         n = taille(t2)
6     res = allouer(n,False)
7     for i in range(n):
8         res[i] = appartient(i,t1) or appartient(i,t2)
9     return(res)
```

Partie II. Représentation par entiers

Question 5 : L'entier $2^n - 1$ est le plus grand entier représentant un sous-ensemble de $\{0, 1, ..., n-1\}$, cet entier représentant le sous-ensemble $\{0, 1, ..., n-1\}$. L'entier 0 est le plus petit et représente l'ensemble vide.

Question 6 : On parcourt le tableau en partant de la fin. On initialise le résultat à 0 (ligne 3). À chaque passage dans la boucle des lignes 4 à 8, on multiplie le résultat par 2 et on ajoute 1 si l'élément $n-1-i$ appartient à l'ensemble S représenté par le tableau t. Au final, on obtient l'entier représentant l'ensemble associé au tableau t en un temps linéaire (un seul parcours de boucle).

```
1  def set2int(t):
2      res = 0
3      n = taille(t)
4      for i in range(n):
5          if t[n-1-i]:
6              res = 1 + 2*res
7          else:
8              res = 2*res
9      return(res)
```

Question 7 : On commence par déterminer la taille du tableau à créer en fonction de n (boucle des lignes 4-6). Si n est compris entre 2^p et $2^{p+1}-1$, le tableau créé aura pour longueur p (sauf dans le cas particulier $n=0$). Ensuite, on crée le tableau résultat (ligne 7) que l'on remplit correctement grâce à la boucle des lignes (10-14). Le temps d'exécution est en $O(\log_2(n))$, les deux boucles conditionnelles étant parcourues au plus $p+1$ fois si n appartient à l'intervalle $[\![2^p, 2^{p+1}]\!]$.

```
1  def int2set(n):
2      taille = 1
3      m = quotient_div_2(n)
4      while m != 0:
5          taille += 1
6          m = quotient_div_2(m)
7      res = allouer(taille,False)
8      m = n
9      for i in range(n):
10         if parite(m) <> 0:
11             res[i] = True
12         m = quotient_div_2(m)
13     return(res)
```

Partie III. Familles, sous-familles, et couvertures

Question 8 : On considère U l'ensemble des étudiants et pour une activité donnée a, l'ensemble F_a des élèves voulant pratiquer cette activité. On considère l'ensemble F de tous les F_a pour toutes les activités a. L'ensemble F constitue bien un recouvrement car chaque élève a choisi au moins une activité. Le problème de l'école est de donner au moins une activité à chaque élève tout en minimisant le nombre d'activités, c'est bien un problème de couverture optimale.

Question 9 : On considère $n=6$ et $F=(\{0,2,4\},\{0,1\},\{2,3\},\{4,5\})$. L'algorithme glouton conduit à choisir F comme sous-famille couvrante alors que la sous-famille couvrante optimale est $(\{0,1\},\{2,3\},\{4,5\})$.

Question 10 : On utilise les fonctions `diff` et `cardinal` agissant sur des entiers. Le résultat de la fonction `reste` n'est autre que le cardinal de l'ensemble $S_1 \setminus S_2$.

```
1  def reste(s1,s2):
2      return(cardinal(diff(s1,s2)))
```

Question 11 : À la ligne 5, on calcule 2^n-1 qui est le nombre représentant l'ensemble $\{0,...,n-1\}$. Les lignes 7 à 9 servent à construire un tableau contenant les puissances successives de 2. Dans la boucle conditionnelle des lignes 11 à 19, on applique l'algorithme glouton. À chaque passage dans la boucle, on détermine le sous-ensemble élément de F qui contient le plus d'éléments de U. Ce sous-ensemble étant trouvé, on ajoute au résultat son indice dans le tableau f (ligne 19) et on supprime ses éléments présents dans U (ligne 18).

```
1  def glouton():
2      global n, f
3      p = taille(f)
4      # calcul de 2^n-1 représentant {0,1,...,n-1}
5      u = set2int(allouer(n,True))
6      # on construit le tableau des puissances de 2.
7      puiss2 = allouer(p,1)
8      for i in range(1,p):
9          puiss2[i] = 2*puiss2[i-1]
10     res = 0
11     while u != 0:
12         c = cardinal(u)
13         # On cherche l'élément de F contenant le plus d'éléments de u
14         for i in range(p):
15             if reste(u,f[i]) < c:
16                 c = reste(u,f[i])
17                 indice = i
18         u = diff(u,f[indice])
19         res += puiss2[indice]
20     return(res)
```

Le coût de « `taille(f)` » (ligne 3) est en $O(p)$ où p est la taille de f, celui du calcul de u (ligne 5) en $O(n)$. Le coût de la première boucle (lignes 8-9) est en $O(p)$. Dans le pire des cas, dans la boucle conditionnelle (lignes 11-19), on supprime un seul élément dans U à chaque passage et donc cette boucle sera parcourue au plus n fois. La boucle interne des lignes 14 à 17 est parcourue p fois et la fonction «reste» ayant un coût en $O(n)$, on obtient un coût global en $O(n^2\,p)$.

Question 12 : Dans la boucle des lignes 5 à 7, on calcule l'entier représentant l'ensemble $\bigcup_{\ell\in int2set(g)} F_\ell$. En supposant que les F_ℓ sont bien tous inclus dans l'ensemble $\{0,...,n-1\}$, il suffit de regarder si le cardinal de l'ensemble calculé est bien égal à n pour avoir un recouvrement.

```
1  def couverture(g):
2      global f, n
3      res = 0
4      G = int2set(g)
5      for l in range(taille(G)):
6          if G[l]:
7              res = union(res,f[l])
8      return(cardinal(res) == n)
```

Question 13 : Pour trouver la solution optimale, on étudie toutes les parties de F (boucle des lignes 10 à 14) et parmi celles qui recrouvrent l'ensemble $\{0,1,...,n-1\}$, on retourne une solution de cardinal minimal. Dans le test des lignes 12 à 14, si g est associé à un recouvrement de $\{0,1,...,n-1\}$ et que son cardinal est strictement inférieur à la meilleure des solutions trouvées auparavant, alors on actualise la solution provisoire « `res` » et le cardinal du meilleur recouvrement provisoire « `CardinalOptimal` ».

```
1  def optimal():
2      global n, f
3      p = taille(f)
4      cardPf = 1
5      # On calcule le nombre de parties d'un ensemble à n éléments,
6      # soit 2^n.
7      for i in range(p):
8          cardPf *= 2
9      CardinalOptimal = cardPf
10     res = cardPf - 1
11     for g in range(cardPf):
12         cardinalG = cardinal(int2set(g))
13         if couverture(g) and cardinalG < CardinalOptimal:
14             CardinalOptimal = cardinalG
15             res = g
16     return(res)
```

Le coût du calcul de 2^p est p. La boucle des lignes 10 à 14 est parcourue 2^p fois. Le coût de la fonction `cardinal` est en $O(n)$ (d'après la solution proposée à la question 1), le coût de la fonction `couverture` étant en $O(np)$ (On calcule la réunion d'au plus p ensembles de cardinal au plus n), on obtient un coût global en $O(np\,2^p)$.