info204 : Science informatique
TD 2 : codage et compression

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/
wiki : http://www.lama.univ-savoie.fr/wiki

Exercice 1 : Application du cours

Question 1. Donnez la représentation en complément à 2 sur 8 bits des nombres :
- $a = 42$,
- $b = -70$,
- $c = -88$,
- $d = -34$.

Calculez $e = a + b$ en utilisant exactement l'algorithme d'addition en base 2. Vérifiez que le résultat est correct.

Calculez $f = d + c$ de la même façon et vérifiez que le résultat est correcte.

Calculez maintenant $d + f$. Qu'en pensez-vous ?

Question 2. Pour les flottants IEEE754 (32 bits) :
- quel est le plus grand nombre réel que l'on peut représenter ?
- quel est le plus petit nombre réel strictement positif que l'on peut représenter ?
- que se passe-t'il si vous ajouter le premier au second ?

Question 3. Ceux qui ont un ordinateur sous la main peuvent essayer

```
a = 49.0
b = 1.0/a
c = b*a

if (1.0 == c):
  print("OK")
else:
  print("Oups...")
```

Pour les autres, que pensez-vous qu'il peut se passer ?

Comment corriger le problème ?

Exercice 2 : Réfléchissons sur la compression...

Question 1. Est-ce qu'un algorithme de compression (`zip` par exemple) peut compresser *tous* les fichiers ?

Question 2. Nous allons montrer la chose suivante :

> *si* `zip()` *est un algorithme de compression quelconque, et si* `d` *est un document, alors soit la suite*
>
> `d, zip(d), zip(zip(d)), zip(zip(zip(d))), ...`
>
> *contient des fichiers arbitrairement gros, soit il existe un n tels que* $\texttt{zip}^n(\texttt{d}) = \texttt{d}$.

Question 3. Pour un texte en ASCII (ou ISO-8859), la taille du fichier en octets est exactement le nombre de caractères utilisés dans le fichier.

Décrivez un algorithme de compression (et de décompression) qui permette de gagne de la place si le fichier en question ne contient que 2 caractères (`.` et `?` par exemple).

Quelle sera la taille d'un ficher compressé (en fonction de la taille du fichier décompressé) ?

Question 4. Est-ce que votre algorithme se généralise facilement à un fichier qui contient 3 ou 4 (ou plus) caractères différents ?

Question 5. Dans la langue française, la lettre `e` apparaît deux fois plus souvent que n'importe quelle autre lettre. (Les fréquences suivantes sont `s`, `a` et `i` à des fréquences similaires.)

Comment pouvons-nous mettre ceci en oeuvre pour essayer de compresser un texte en français ?

Question 6. L'algorithme de compression de Huffman fonctionne de la manière suivante : on trie les caractères par ordre décroissant de fréquence, par exemple

`(i,7.1%) (a,7.2%) (s,7.5%) (e,14%)`

On prend les deux caractères les moins fréquents, et on les *ajoute* en mettant la somme de leurs fréquences :

`(s,7.5%) (e,14%) (0i,1a,14.7%)`

Et on continue :

`([0i,1a],14.7%) ([0s,1e],21.5%)`

et encore :

`([00i,01a,10s,11e],32.2%)`

À chaque étape, on ajoute un `0` et `1` devant les lettres...

À la fin, on obtient ainsi un *code* pour chaque lettre. Dans notre cas :

- `00` pour `i`,
- `01` pour `a`,
- `10` pour `s`,
- `11` pour `e`.

Appliquez l'algorithme pour trouver les codes sur un texte contenant :

`a` : 55, `b` : 14, `c` : 12, `d` : 32, `e` : 10, `f` : 7

Question 7. Pourquoi (et comment) peut-on décoder un texte compressé de cette manière ?

Remarque : ce codage est *optimal* pour la compression "caractère à caractère".