1ère Année ENSEIRB option Telecom

Algorithmique et structure de données

Corrigé du devoir d'entraînement

1 Permutations

Une permutation α de S_n est une suite de n nombres entiers

$$\alpha = a_0, a_1, \ldots, a_{n-1}$$

tous différents et compris entre 0 *et* $n-1$.

1. *Une descente dans une permutation est un entier i tel que $a_i > a_{i+1}$*

   Algorithme qui étant donnée une permutation représentée par un tableau `alpha` affiche le nombre de ses descentes.

```
res = 0;
for (int i = 1; i < n; ++i)
     if (alpha[i] < alpha[i-1])
         ++res;
afficher(res);
```

   Attention à ce que l'indice d'accès au tableau ne sorte pas des limites.

2. *Une inversion dans une permutation est un couple i, j tel que $i < j$ et $a_i > a_j$.*

   Algorithme qui affiche le nombre d'inversions d'une permutation représentée par son tableau.

```
res=0;
for (int i = 0; i < n-1; ++i)
     for (int  j = i+1; j <n; ++j)
         if (alpha[j] < alpha[i]) res++;
afficher (res);
```

3. Un algorithme qui vérifie si un tableau de taille n représente bien une permutation, c'est à dire s'il contient bien tous les nombres entiers compris entre 0 et $n-1$. On fait une boucle qui verifie que tous les `alpha[i]` sont compris entre `0` et `n-1`; un tableau `vu` dont tous les élements sont initialisés à `false`, permet de savoir si on a déjà vu `alpha[i]`. La varaible `u` prend la valeur `false` dès que l'on a trouvé un `alpha[i]` hors de l'intervalle `0... n-1` ou qui apparaît deux fois. Le test sur `u` dans la boucle for permet de sortir de cette boucle dans ce cas.

```
// tableau de booleens vu
 u = true;
 for (i = 0; i < n ; ++i)
      vu[i] = false;
 for (i = 0; i < n && u; ++i) {
     j = alpha[i];
     if (j < 0 || j >= n ||  vu[j])
          u = false;
     else  vu[j] = true;
 }
 if (u) afficher("est une permutation");
 else afficher("n'est pas une permutation");
```

4. *Un cycle dans une permutation est une suite de nombres $i_1, i_2, \ldots, i_p$ tels que $\alpha(i_j) = i_{j+1}$ pour $j = 1, \ldots p-1$ et $\alpha(i_p) = i_1$. .*

   Un algorithme qui détermine le nombre de cycles d'une permutation donnée par un tableau. On utilise un tableau auxiliaire `vu` contenant des booléens tous initialisés à `false`, et tel que `vu[i]` est mis à `true` quand on considère l'élement `i` dans l'ensemble des cycles parcourus.

```
for (i = 0; i < n ; ++i)
      vu[i] = false;
res = 0;
for (i = 0; i < n; ++i)
    if (!vu[i]) {
        ++res;
        vu[i] = true;
        for (int  j = alpha[i]; j != i; j = alpha[j])
            vu[j] = true;
     }
 affiche(res);
```

2 Rendre la monnaie

On vous propose d'écrire un algorithme permettant d'obtenir la suite des billets totalisant une somme donnée (dont on suppose qu'elle est un multiple de 10*). Les espèces disponibles sont des billets de 50, 20 et 10 euros. Le principe est de donner le billet de valeur la plus grande possible inférieure ou égale à la somme à rendre et de poursuivre la même stratégie avec la somme restante jusqu'à ce que la somme restante soit nulle.*

1. Algorithme qui affiche le nombre de billet de chaque espèce:

```
while (s != 0) {
     if (s >= 50) {
```

```
        s = s - 50;
        n1++;
    }
    else if (s >= 20 ){
        s = s -20;
        n2++;
    }
    else {
        s = s - 10;
        n3++;
    }
}
Afficher (n1, "billets de 50 euros);
Afficher (n2, "billets de 20 euros);
Afficher (n3, "billets de 10 euros);
```

2. *Déterminer le nombre de soustractions effectuées par l'algorithme ; on donnera une formule faisant intervenir des divisions entières par les nombres 50, 20 et 10.*

   Soit q_1 le quotient entier de s par 50, $q2$ celui de $s - 50q_1$ par 20, et q_3 la valeur de $\frac{s-50q_1-20q_2}{10}$ le nombre de soustractions effectuées est $q_1 + q_2 + q_3$ qui inférieur ou égal à

   $$\frac{s}{50} + 3$$

3. Le nombre de billets donné par l'algorithme précédent est le plus petit possible. Pour cela on considère les 5 possibilités de la valeur du reste de la division par 50 de la somme s.

   - Si la somme est un multiple de 50 alors l'algorithme ne donne que des billets de 50 et il n'est pas possible de faire mieux,
   - Si la somme s est égale à $50a + 10$, l'algorithme donne a billets de 50 et 1 billet de 10.
   - Si la somme s est égale à $50a + 20$, l'algorithme donne a billets de 50 et 1 billet de 20.
   - Si la somme s est égale à $50a + 30$, l'algorithme donne a billets de 50 1 billet de 20 et 1 billet de 10.
   - Si la somme s est égale à $50a + 40$, l'algorithme donne a billets de 50 et 2 billets de 20.

   Toute autre façon de constituer ces sommesavec des billets de 50, 20 et donnera un nombre de billets plus grand

4. Généralisation de l'algorithme de détermination du nombre de billets à rendre au cas où les valeurs de billets disponibles sont en nombre quelconque et figurent dans un tableau à valeurs décroissantes `val[0] > val[1] > ... > val[k-1]`.

```
k = 0;
while (s > 0) {
   while (val[k] > s) k++;
   tab[k]++;
   s = s- val[k];
}
Afficher(tab);
```

5. *Montrer qu' il existe des valeurs de billets et une somme à rendre pour lesquels cet algorithme ne donne pas le nombre minimum de billets ou à rendre.*

   Il y a en effet des cas où cet algorithme ne donne pas le plus petit nombre de billets à rendre, par exemple si on dispose de billets de 50, 20, 12, 1, et que la somme à rendre est de 24, l'algorithme donnera 5 billets à rendre (1 de 20 et 4 de 1) alors que l'on peut faire avec 2 billets de 12.