Travaux Dirigés IF122 n°12

deug

Révisions

▸ **Exercice 1.** Le but de cet exercice est d'afficher tout les mots d'une certaine longueur, construit sur 0,1 et ayant autant de 0 que de 1.

1. Que fait le programme suivant :

```
void f(){
 int i1,i2;
 for(i1=0;i1<=1;i1++){
     for (i2=0;i2<=1;i2++) {
          cout << i1 << i2 << endl;
     }
 }
}
```

✁ -

et bien il affiche tout les mots de 2 lettres fait de 0 et de 1

✁ -

2. Ecrivez une procedure qui affiche tout les mots de 4 lettres ayant autant de 0 que de 1;

✁ -

On introduit 2 boucles de plus. On peut remarquer que si j'ai autant de 0 que de 1, alors j'en ai 2 !!!!!!!!!!

```
void f(){
 int i1,i2,i3,i4;
 for(i1=0;i1<=1;i1++){
     for (i2=0;i2<=1;i2++) {
        for (i3=0;i3<=1;i3++) {
            for (i4=0;i4<=1;i4++) {
        if (i1+i2+i3+i4==2) cout << i1 << i2 << i3 << i4 << endl;
            }
        }
     }
 }
}
```

✁ -

3. Dans quelle mesure peut on généraliser ?

✂ -

Pas trop comme ca, a priori il faut compliquer.

✂ -

4. Le problème est à présent le suivant : On demande à l'utilisateur de fixer la longueur des mots, puis le programme doit tous les afficher. On choisit de modéliser les mots par des listes. On suppose que l'on dispose des procédures suivantes : **ajoute_tete(liste &l,int i), ajoute_queue(liste &l,int i), efface_tete(liste &l), efface_queue(liste &l), affiche_ordre(liste l), affiche_ordre_inverse(liste l)**. Proposez une solution. Si vous en avez plusieurs, comparez en la complexité.

✂ -

```
void affiche(int rest0, int rest1,liste m){
 liste m1,m2;
 if (rest0+rest1==0) affiche_ordre(m);
 else {
 if (rest0) {m1=ajoute_tete(m,0); affiche(rest0-1,rest1,m1);}
 delete m1;
 if (rest1) {m2=ajoute_tete(m,1); affiche(rest0,rest1-1,m2);}
 delete m2;
 }
}
void main(){
int L;
liste m=NULL;
cout <<  ``entrez la longueur des mots'';
cin >> L;
affiche(L/2,L/2,m);
}
```

Alors il y a des choses de cachées ici : si on avait fait le choix d'ajouter en queue, alors il aurait fallut penser a supprimer ce qui a ete mis en plus apres chaque appel recursif. L'affichage dans l'ordre ou pas est indifferents, c'est une histoire de convention, mais ca doit amener la question : dans quel ordre sont generes les mots.

Lors de la correction vous pouvez proposer aussi la seconde solution suivante. Se pose a lors le Pb de la complexite. Dans la seconde on génere les 2^L mots (il faut y ajouter la complexite du test) dans la première on ne génere que les $C_L^{L/2}$.

```
void affiche(int longueur,liste m){
 liste m1,m2;
 if (longueur==0 && difference_01(m)==0) affiche_ordre(m);
 else {
 m1=ajoute_tete(m,0); affiche(longueur-1,m1);
 delete m1;
 m2=ajoute_tete(m,1); affiche(longueur-1,m2);
 delete m2;
 }
}
int difference_01(liste m){
 if (m==NULL) return 0;
 return ( difference_01(m->suivant) + (2* m->val -1)); // pour rigoler
}
```

```
void main(){
int L;
liste m=NULL;
cout <<  ‘‘entrez la longueur des mots’’;
cin >> L;
affiche(L/2,L/2,m);
}
```

✄ -

5. Il est possible d’obtenir un résultat similaire sans utiliser la récursion. Proposez une méthode originale, sans rentrer dans tout les détails. (Que pensez vous de la representation binaire des nombres, et des opérateurs $>>$ et $<<$?

✄ -

Allez, je propose de faire la chose suivante :
1) on construit une liste de longueur L constituée de 0’s
2) on la voit comme un nombre binaire, et on passe au suivant en faisant + 1 ce qui permet de tous les enumerer.
3) on utilise un test le long de l’enumeration pour savoir si le mot est conforme.

✄ -

6. Implementez la méthode précédente

✄ -

Il est a rapeller que si x est un entier, x ¡¡ 2 et x ¿¿ 2 sont les decalages de bit a gauche et a droite.

```
void main(){
int L,m;
cout <<  ‘‘entrez la longueur des mots’’;
cin >> L;
for(m=0;m<2^L;m++){ if (count_01(m,L)==0) cout << m << endl;}
}

int count_01(int m,int L){
 int aux,rep=0,i;
 for(i=0;i<L;i++){
  aux=m>>i;
  aux=aux -(aux>>1)<<1;
  if (aux==1) rep++; else rep--;
 }
return rep;
}
```

✄ -

▸ **Exercice 2.** La hauteur de l'arbre vide est 0. La hauteur d'un arbre est égale au maximum des hauteurs de ses fils plus 1. Un arbre est complet si chacun de ses noeuds a des fils de même hauteur.

1. quelle est la hauteur d'une feuille ?
2. dessiner un arbre binaire complet de hauteur 2.
3. Combien de feuilles y a t'il dans un arbre binaire complet de hauteur h ?
4. Combien de noeuds ?

✂ -

Un arbre de hauteur $h \neq 0$ a 2^{h-1} feuilles : $F_1 = 1$ et $F_{h+1} = F_h + F_h$ (les feuilles pour chacun de ses fils + le fait que les fils sont complets)

Pour les noeuds : $N_{h+1} = N_h + F_{h+1}$ avec $N_0 = 0$ c'est que $N_h = \sum_{i=1}^{h} 2^{i-1} = \sum_{i=0}^{h-1} 2^i = 2^h - 1$

✂ -

▸ **Exercice 3.** Que fait le programme suivant :

```
int f(arbre &a) {
    if (a==NULL) {
       a=new (noeud);
       a->val =1;
       a->gauche=NULL;
       a->droite=NULL;
       return 1;
    }
    else{
     a->val =  f(a->gauche) + f(a->droite);
     return a->val;
   }
}
void main(){
arbre a=arbre_aleatoire();
int i,n,x;
cout << ``donner moi une valeur ``;
cin >> n;
 for (i=0;i<n;i++){
 cout << f(a);
 }
}
```

✂ -

Il fait deux choses : il agrandit l'arbre en lui rajoutant une couche de feuilles (c'est le printemps), et chaque noeud est informé du nombre de feuilles que contient son sous arbre (large)

Passez directement à l'exercice qui reparle de 3. En leur disant de modifier le code. (Je l'ai placé plus loin pour ne pas leur donner qq indication que ce soit.

Bon, pour tester on peut commencer par a=NULL.

✂ -

▸ **Exercice 4.**
Une liste chainée circulaire est une suite de cellules, exactement comme une liste chainée, sauf que le successeur du dernier element est le premier element.

1. Proposez un type de donnée pour les listes chainées circulaires.

2. Ecrivez une fonction qui effectue l'union de deux listes chainées circulaires.

✂ -

```
noeud * get_last(liste_c a){
     noeud *last=a;
     if (a==NULL) {cout << ``pb''; return NULL;}
     while (last->suiv != a) {last=last->suiv;}
     return last;
}

liste_c fusion(liste_c a, liste_c b){
   if (a==NULL) return b;
   if (b==NULL) return a;
   *cell aux= get_last(a);
   aux->suiv=b;
   aux= get_last(b);
   aux->suiv=a;
   return a;  // ou aux, ou b
}
```

✂ -

▸ **Exercice 5.** La procédure suivante a pour arguments $max1, max2$ qui sont passé en reference. On veut que l'appel de cette procédure modifie ces variables pour qu'elles contiennent les deux plus grandes valeurs du tableau A. (ces deux valeurs peuvent être égales). Le dernier paramètre représente la taille du tableau. Pour éviter de nombreuses solutions, (et pour en avoir de simples) on vous impose de n'utiliser que l'equivalent d'une boucle, au plus. Donner une version iterative et une version recursive.

```
void plusgrandes (int& max1, int& max2, int A[], int n)
```

✂ -

```
void plusgrandes (int& max1, int& max2, int A[], int n){
  int i;
  if (n<2) cout << ``question idiote'';
  else{
      if (A[0] < A[1]) {
         max1=A[1]; max2=A[0];}
      else {
         max1=A[0]; max2=A[1];}

      for (i=2;i<n;i++){   // on commence a 2
```

```
         if (A[i]>max2){  // remarquez que je compare a max2 et pas a max1
            if (A[i]<max1) max2=A[i];
            else {
            max2=max1;max1=A[i];
            }
         }
      }
}
```

Moins d'une boucle ca va etre dur, on essaye de la cacher :

```
void plusgrande (int& max1, int& max2, int A[], int n){
 if (n<2) cout << ‘‘question idiote’’;
 else{
   if (n==2){
      if (A[0] < A[1]) {
         max1=A[1]; max2=A[0];}
      else {
         max1=A[0]; max2=A[1];}
   else{
      plusgrande(max1,max2,A+1,n-1);
      if (A[0] >max2){  // remarquez que je compare a max2 et pas a max1
         if (A[0]<max1) max2=A[0];
         else {max2=max1;max1=A[0];
         }
      } // sinon on ne fait rien
   }
}
```

✂ -

▸ **Exercice 6.** Modifier le programme de l'exercice 3 de sorte que chaque noeud contienne à présent le nombre de noeuds internes du sous arbre dont il est racine.

✂ -

```
a->val =1; ----> a->val =0;
return 1;  ----> return 0;

a->val = 1 + f(a->gauche)+ f(a->droite);
```

✂ -

▸ **Exercice 7.** Supposez qu'on veuille réaliser les deux opérations find(a) et delete_min sur un ensemble S, qui correspondent à la recherche d'un élément a et à la suppression du minimum.

Pour les structures de données suivantes indiquez si le temps d'éxécution des deux opérations find et delete_min est constant, logarithmique, linéaire, exponentiel. La réponse doit être aussi précise que possible (par exemple, si le temps est logarithmique, la réponse ¡¡linéaire¿¿ est considérée comme étant imprécise).

1. liste non triée, triée

2. tableau, tableau trié
3. tas-min,
4. arbre 2-3-4
5. table de hachage

✂ -

Il y une remarque particuliere a faire pour les tableau tries : il est precise que l'on ne s'interesse qu'a find et delete. Cette remarque permet d'enrichir le tableau avec l'indice i du plus petit element. Cette ¡¡ruse¿¿ permet de ne pas faire le decalage correspondant a la suppression. Ainsi la suppression du min se fait en temps constant : on ne dit que $i++$, la recherche se fait de facon dichotomique sur le reste et est donc en log.

la suppresion pour les arbre 2-3-4 n'est pas facile, et la justification a du etre rapelle en cours.

	find	del_min
liste non triée	n	n
liste triée	n	1
tableau	n	n
tableau trié	log	n ou 1
tas-min	n	log n
arbre 2-3-4	log	log ?
table de hachage	1	n

✂ -

▸ **Exercice 8.** Ecrivez une fonction qui échange l'un des minimums et l'un des maximums dans un arbre.

✂ -

```
arbre max(arbre a){
   if (a==NULL) {cout << ``erreur''; return NULL;}
   arbre x=a,y=a,rep=a;
   if (a->gauche != NULL) {
      x=max(a->gauche);
      if (x->val > a->val) rep = x;
   }
   if (a->droite != NULL) {
      y=max(a->droite);
      if (y->val > x=rep->val) rep=y;
   }
   return rep;
}
arbre min(arbre a){
   if (a==NULL) {cout << ``erreur''; return NULL;}
   arbre x=a,y=a,rep=a;
   if (a->gauche != NULL) {
      x=min(a->gauche);
      if (x->val < a->val) rep = x;
   }
   if (a->droite != NULL) {
      y=min(a->droite);
```

```
      if (y->val < x=rep->val) rep=y;
   }
   return rep;
}
arbre echange(arbre a){
 arbre m,M;
 m=min(a);
 M=max(a);
 int aux=m->val;
 m->val=M->val;
 M->val=aux;
}
```

✂ --

▸ **Exercice 9.** On suppose que la racine de l'abr a a bien deux fils. Ecrire une procédure qui échange le sous arbre de racine min avec le sous arbre de racine max. Comment faire dans le cas d'un arbre ?

✂ --

```
void echange(arbre a){
 cell *pere_min=a;
 cell *pere_max=a;
 while (pere_min->gauche->gauche) pere_min=pere_min->gauche;
 while (pere_max->droite->droite) pere_max=pere_max->droite;

 cell *aux=pere_min;
 pere_min=pere_max;
 pere_max=aux;
}
```

Dans le cas d'un arbre : cela n'a pas de sens d'effectuer cette operation. En effet, si l'un est sous arbre de l'autre, l'echange donne des choses aberantes non ?

✂ --

▸ **Exercice 10.** Donner une version iterative d'un programme