

Examen d'Algorithmique - session 2010

Durée : 2 heures

Formation Ingénieurs CNAM

Ce sujet comporte 4 pages et 2 parties indépendantes. On s'attachera à soigner la présentation du code afin qu'il soit le plus lisible possible. Il ne suffit en aucun cas d'écrire le code d'une fonction, il faut expliquer (i.e. commenter) les choix faits pour l'implantation. Il est de plus indispensable de respecter les notations données dans l'énoncé.

1 Echauffement

On rappelle la structure de données définie pour représenter les listes d'entiers ainsi que les opérations de base `vide` et `cons`. Afin de pouvoir tester notre implantation, on programme également une fonction d'affichage `print`.

```
#include<stdio.h>

typedef struct liste
{
    int x;
    struct liste *suivant;
} Sliste, *Liste;

Sliste *vide()
{ return (NULL); }

Sliste *cons(int v, Sliste *l)
{
    Sliste *m=(Sliste *)malloc(sizeof(Sliste));
    m->x=v;
    m->suivant=l;
    return m;
}

void print(Sliste *l)
{
    while (l!=NULL)
    {
        printf("%d ",l->x);
        l=l->suivant;
    }
    printf("\n");
}
```

Question 1 Programmer la fonction `plus1` (qui ajoute 1 à chaque élément de la liste). On procédera de manière itérative (en utilisant une boucle `while`) avec des effets de bord (le type de retour de la fonction demandée est `void`).

```
void plus1 (Sliste *l)
{ ... }
```

Question 2 Programmer de manière récursive une fonction `somme` qui fait la somme de tous les éléments d'une liste.

```
int somme(Sliste *l)
{ ... }

int main()
{ ... }
```

Question 3 Ecrire un programme de test dans la fonction `main` qui produit la sortie suivante :

```
user@computer$ ~/test-listes
3 6 8
somme = 17
4 7 9
somme (+1)= 20
user@computer$
```

2 Problème : arbres binaires parfaits partiellement ordonnés

Un arbre binaire est dit *parfait* lorsque tous ses niveaux sont complètement remplis, sauf éventuellement le dernier niveau et dans ce cas les noeuds (qui sont des feuilles) du dernier niveau sont groupés le plus à gauche possible. L'arbre ci-dessous est un arbre parfait. Un arbre

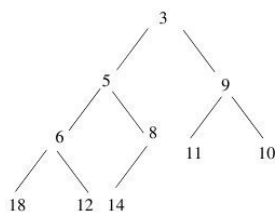


FIGURE 1 – Exemple de tas

binaire parfait de taille n peut se représenter de façon très compacte dans un tableau de taille au moins égale à $n + 1$ (la première case n'est pas utilisée). La représentation est la suivante :

La racine de l'arbre est en $T[1]$ et si un noeud se trouve en $T[i]$, son fils gauche est en $T[2*i]$ et son fils droit en $T[2*i+1]$.

Question 4 Dessiner un tableau de taille 10 permettant de représenter l'arbre binaire de la figure 1.

Question 5 Dessiner l'arbre binaire parfait qui correspond au tableau suivant (la première case - d'indice 0 - est vide) :

	3	12	7	18	16	22	13
--	---	----	---	----	----	----	----

On considère le canevas de programme de la figure 2. Vous répondrez aux questions suivantes en utilisant les structures de données et fonctions définies dans ce morceau de programme.

Question 6 Expliquer la structure de données et les différentes fonctions définies dans la figure 2 et leurs rôles. On précisera notamment le sens des arguments ainsi que les préconditions utiles qu'il pourrait être nécessaire de rajouter dans le code.

Un arbre binaire parfait est dit *partiellement ordonné* si l'étiquette de tout noeud est inférieure à celle de ses fils. Le minimum de l'ensemble représenté par un tel arbre est donc situé la racine. Un tel est appelé *un tas*.

Question 7 L'arbre dessiné à la question 5, qui est parfait, vérifie-t-il la propriété d'être partiellement ordonné ?

Dans la suite, on travaille avec des arbres binaires parfaits partiellement ordonnés (aussi appelés "tas").

2.1 Ajouter un élément dans le tas

On commence par rajouter l'élément x à insérer comme feuille au dernier niveau de l'arbre puis on échange l'étiquette du noeud portant x avec celle de son père jusqu'à ce que, pour respecter la structure d'arbre binaire parfait partiellement ordonné, x ne soit pas inférieur à l'étiquette du père du noeud qui le porte.

Question 8 Faire un dessin des étapes successives de l'insertion du nombre 4 dans l'arbre de la figure.

Question 9 Programmer une fonction **insertion** qui réalise cette opération

```
void insertion(tas *x, int i)
{ ... }
```

2.2 Supprimer l'élément minimum du tas

L'idée est de supprimer la dernière feuille du dernier niveau après avoir recopié son étiquette à la racine, puis, toujours pour respecter la structure d'arbre binaire parfait partiellement ordonné, de faire redescendre cette valeur en la comparant au contenu de ses fils, par un processus inverse de celui de l'insertion.

Question 10 Faire un dessin des étapes successives à réaliser lors de la suppression du minimum dans l'arbre donné en exemple dans la figure. Lors de la descente d'une valeur dans l'arbre, on privilégiera la branche avec l'étiquette minimale lorsqu'on a le choix du côté de l'arbre où faire descendre l'élément courant.

Question 11 Programmer une fonction **suppression_min** qui supprime le minimum (i.e. la racine) dans un arbre binaire parfait partiellement ordonné.

```
void suppression_min(tas *x)
```

2.3 Un programme de tri

Question 12 Expliquer comment les opérations programmées précédemment pourraient être utilisées pour trier n nombres par ordre croissant. On supposera que les données fournies en entrée le sont sous la forme d'un tableau de taille n .

Question 13 Programmer un tel algorithme avec les fonctions construites jusqu'ici.

```

#include<stdio.h>
#define N 10

typedef struct stas
{ int t[N];
  int n;} tas;

int racine(tas x, int i)
{return i;}

int etiquette(tas x, int i)
{return x.t[i];}

int is_vide(tas x, int i)
{return (i==0);}

int ag(tas x, int i)
{
  if (2*i<=x.n) return (2*i); else return 0;
}

int ad(tas x, int i)
{
  if(2*i+1<=x.n) return (2*i+1); else return 0;
}

int pere(tas x, int i)
{return (i/2); }

void print(tas x, int i)
{
  if (!is_vide(x,i))
  {
    print(x,ag(x,i));
    if (i!=0) printf("%d ", etiquette(x,i));
    print(x,ad(x,i));
  }
}

```

FIGURE 2 – Implantation des opérations de base sur les tas