TD 5

Files de priorités et tris par tas

Préliminaires

Nous étudions ici la structure abstraite de *minimier*, et son utilisation pour représenter une file de priorité ou pour trier des éléments selon une clé (on parle alors de*tris par tas*). Cette structure est un arbre binaire et soit e une position dans l'arbre nous utiliserons les fonctions *pere(m,e)*, *filsG(m,e)*, *filsD(m,e)* pour désigner les positions du père de e dans l'arbre, et des fils gauche et droit de e. Les positions sont totalement ordonnées et nous noterons *rac* la position du premier élément, *dernier* celle du dernier élément, *succ(e)* le successeur de la position e dans cet ordre, *pred(e)* le prédécesseur de e. On utilisera de plus une fonction *plusGrand(e,f)* qui renvoie vrai si la position e est supérieure à la position f dans cet ordre.

Les éléments sont rangés dans l'arbre en suivant cet ordre : la racine est *rac*, puis les successeurs sont rangés par niveaux de gauche à droite. Tous les éléments ont une *clé* (par exemple un entier) et satisfont tous la propriété de *dominance* suivante : dans un minimier la clé d'un élément est toujours plus petite ou égale à celle de ses fils [1].

On supposera dans ce qui suit que *rac* est constant alors que *dernier* dépend du nombre d'éléments du minimier. Dans ce qui suit un minimier sera représenté par un tableau m d'éléments et la position *dernier* du dernier élément.

On notera *echange(m,e,f)* la fonction qui échange les éléments de position e et f dans l'arbre. Cette fonction est utilisée en particulier dans les deux principales opérations de mises à jour :

- *maintientMinimierBas(m, r, dernier)* qui est utilisée quand la clé de l'élément r a peut être augmenté. il faut alors descendre l'élément en r par échanges successifs jusqu'à ce que la propriété de dominance soit satisfaite.
- *maintientMinimierHaut (m, d)* qui est utilisée quand la clé de l'élément d a peut être diminué. il faut alors remonter l'élément en d par échanges successifs jusqu'à ce que la propriété de dominance soit satisfaite.

L'insertion d'un nouvel élément *el* se fait en agrandissant le minimier m, en mettant en dernière position le nouvel élément, puis en faisant une mise à jour du minimier. La fonction d'insertion s'écrit alors *insere(m, dernier, el)* et renvoie *succ(dernier)*.

La suppression de l'élément à la racine se fait en le remplaçant par le dernier élément, en réduisant le minimier et en faisant une mise à jour du minimier. La fonction de suppression s'écrit alors *supprimeRac(m,dernier)* et renvoie *pred(dernier)*.

Exercice 1 (maintenir un minimier.)**.**

Voici deux exemples d'arbres binaires qui ne sont pas des minimiers car la propriété de dominance n'est pas respectée pour un des noeuds. Les noeuds sont donnés dans l'ordre à partir de la racine :

- $cle(rac) = 1., 4., 3., 8., 0., 5., 4.$
- $cle(rac) = 18., 4., 3., 8., 9., 5., 4.$

Donner à chaque fois, en notant n la position de l'élément fautif ainsi que la suite d'instructions permettant de maintenir le minimier, et dessiner les deux minimiers mis à jour.

Exercice 2 (L'ordre des positions et les fonctions associées.)**.**

La manière classique de présenter l'implémentation d'un minimier est de prendre rac=1 et de considérer l'ordre direct *: les noeuds sont numérotés 1 , 2, ... , dernier. Cependant il peut être utile de conserver l'ordre direct et de prendre $rac \neq 1$, en particulier lorsqu'on utilise un langage comme le C où les tableaux sont indexés à partir de 0. Plus généralement ceci permet de ranger les éléments du minimier à un endroit quelconque dans un tableau. Il peut être également utile de le ranger en ordre* inverse *c'est à dire dans l'ordre rac, rac-1, ... , dernier. En particulier nous*

[1] En inversant la relation d'ordre on obtient un *maximier*.

verrons plus loin que l'on peut ainsi ranger un nombre constant n d'éléments dans deux files de priorités représentées par un seul tableau de n éléments.

On sait que dans un minimier en ordre direct avec $rac = 1$*, on a :*

- $pere(m, e) = \lfloor e/2 \rfloor$
- $filsG(m, e) = 2e$ *;* $filsD(m, e) = 2e + 1$
- $succ(e) = e + 1$ *;* $pred(e) = e - 1$
- $plusGrand(e, f) = e > f$

1. *Ecrire* $pere(m, e), filsG(m, e), filsD(m, e)$ *pour* $rac = 0$*, et plus généralement pour* $rac \neq 1$*, quand l'ordre est* direct*. Dessiner l'arbre des positions en ordre direct pour* rac = *0 et* dernier=5 *ainsi que pour* rac = *5 et* dernier=10*.*
2. *Ecrire* $succ(e), pred(e), plusGrand(e, f), pere(m, e), filsG(m, e), filsD(m, e)$ *pour* $rac = n$ *quelconque lorsque l'ordre est* inverse*. Dessiner l'arbre des positions en ordre inverse pour* rac = *5 et* dernier=0*.*

Exercice 3 (Insertion d'un élément et *plantation* d'un minimier)**.**
Le but de cet exercice est de ranger un tableau t de n éléments dans un minimier à partir de rac. Il s'agit à la fois de ranger les éléments et de s'assurer de ce que la propriété de dominance soit respectée pour tous les noeuds de l'arbre. Il y a deux manières de procéder que nous allons expérimenter, mais d'abord nous introduisons la fonction d'insertion *d'un élément dans le minimier.*

1. *Une opération primitive est l'insertion d'un nouvel élément el dans un minimier. On la note* insere(m, dernier, el)*.On opère de la manière suivante : le minimier est agrandi en modifiant* dernier*, l'élément el est placé à la position* dernier *dans le tableaum, enfin on fait remonter l'élément el en utilisant la fonction* maintientMinimierHaut*. Insérer dans le minimier* $cle(rac) = 0., 1., 3., 8., 4., 5., 4.$ *un élément de clé* 2 *et donner la suite d'instructions utilisée pour cela.*
2. *La première manière de planter un minimier est d'insérer successivement dans le minimier m vide les* taille *éléments du tableau t,. La fonction se note* plante(t,n, m) *et renvoie la position du dernier élément. Donner la suite d'instruction permettant de planter dans un minimier le tableau constitué des éléments 10. 9. 8. 7., décrire l'ensemble des échanges ayant lieu et donner le minimier obtenu.*
3. *La deuxième manière de planter un minimier est de recopier d'abord dans le minimier vide m les n éléments du tableau t, puis d'appliquer* maintientMinimierHaut *d'abord à* pere(dernier) *puis en allant vers l'arrière en l'appliquant à l'élément précédent* pred(pere(dernier)) *et ainsi de suite jusqu'à la racine. La fonction se note* planteBis(t, n,m) *et renvoie la position du dernier élément. Donner la suite d'instruction permettant de planter dans un minimier le tableau constitué des éléments 10. 9. 8. 7., décrire l'ensemble des échanges ayant lieu et donner le minimier obtenu.*
4. *Donner l'ordre de grandeur asymptotique des fonctions* maintientMinimierHaut(m,dernier) *et* maintientMinimierBas(m,rac) *si m est un minimier de n éléments.*
5. *Comparer les complexités de* plante(t,n, m) *et* planteBis(t, n, m)*..*

Exercice 4 (tri par minimier)**.**
Le tri en ordre croissant par minimier d'un tableau t de n éléments s'effectue de la manière suivante :

a. *Les éléments du tableau à trier sont rangés dans un minimier m. L'indice i est initialisée au début du tableau t.*

b. *La racine est rangée en* $t[i]$*.*

c. *La racine est supprimée du minimier, la structure du minimier, réduit d'un élément, étant maintenue.*

d. *Si le minimier n'est pas vide, on recommence l'étape b en incrémentant i .*

On s'intéressera d'abord à la fonction supprimeRac(m, dernier) *qui renvoie* pred(dernier) *et qui maintient le minimier m.*

1. *Décrivez l'exécution de* supprimeRac(m, dernier) *sur le minimier* cle(rac) = 7., 8., 9. 10..
2. *Décrivez l'exécution de* triParMinimier(t,n) *sur le minimier* cle(rac) = 7., 8., 9. 10..
3. *Quelle est l'ordre de grandeur du nombre d'échanges effectués dans* triParMinimier(t,n) *?*

Exercice 5 (Implémentation d'un Minimier)**.**
Soit un un type element_t *représenté par une structure contenant un champ* cle*. Un minimier rangé en ordre direct dans un tableau à partir d'une constante rac est représenté par le tableau m d'element_t et la position* dernier *du dernier élément dans le tableau. Le type élément utilisé et le type minimier seront déclarés comme suit :*

```
typedef struct Element {
        int cle;
        int pid;
        int priorite;
        int charge;
} element_t;
typedef element_t minimier[N];
```

1. *Implémenter les fonctions de gestion d'un minimier dont les en-têtes suivent. On utilisera pour cela les fonctions* pere(m,e), filsG(m,e), filsD(m,e), succ(e), pred(e), plusGrand(e,f)*.*

```
void maintientMinimierBas(minimier m, int r, int dernier);
void maintientMinimierHaut ( minimier m,  int d);
int insere (minimier m, int dernier, element_t el);
int plante (element_t tab[], int n,  minimier m);
int planteBis (element_t tab[], int n, minimier m);
int supprimeRac(minimier m, int dernier);
void triParMinimier(element_t t[], int n);
void afficheMinimier(minimier m, int dernier);
/* nombre d'elements entre les positions e et f dans le minimier
   y compris e et f  */
int tailleIntervalle(int e, int f);
```

2. *On supposera dans la suite que ces fonctions gèrent un minimier en ordre direct. On notera* pere_I(m,e)filsG_I(m,e), filsD_I(m,e) , succ_I(e), pred_I(e), plusGrand_I(e,f), maintientMinimierBas_I, ... , afficheMinimier_I *les fonctions correspondantes permettant de gérer un minimier en ordre inverse. On note* rac *la variable globale représentant la position de la racine dans un minimier en ordre direct et* rac_I *celle se rapportant à un minimier en ordre inverse. Suffit-il de transcrire vos fonctions de la question précédente en remplaçant* rac *par* rac_I*,* pere *par* pere_I*,...,* plusGrand *par* plusGrand_I *pour obtenir les fonctions correspondantes du cas inverse ?*

Exercice 6 (Gestion d'un Biprocesseur)**.**

Nous allons ici simuler la gestion d'un biprocesseur. Les processus arrivent alternativement aux files d'attente f_0 et f_1 des deux processeurs P_0 et P_1. Chaque processeur traite le processus à sa racine, ce qui diminue la charge de celui-ci d'une constante c. Si la charge du processus est nulle celui-ci est retiré de la file d'attente sinon son attente *augmente et il recule dans la file d'attente du processeur. Lorsque la différence du nombre de processus présents dans les deux processeurs est supérieure à 2 (la répartition des processus est dite* déséquilibrée*) le processus à la racine de la file d'attente du processeur le plus chargé est supprimé de cette file et inséré dans la file du processeur le moins chargé.*

Les deux files sont représentées par deux minimiers rangés dans le même tableau m de taille N. La file f_0 est rangée en ordre direct avec $rac = 0$ et la file f_1 en ordre inverse avec $rac_I = N - 1$. AInsi on peut avoir en tout N processus dans les deux files.

1. *Ecrire la fonction de prototype* int insertionProcessus(minimier m, int dernier, int valPid, int valPriorite, int valCharge) *qui insère un processus de pid* valPid*, de priorité* valPriorite *et de charge* valCharge *dans la file* f_0 *et renvoie la position du dernier élément de cette file : le dernier dans le minimier rangé dans* m *en ordre direct. (*f_0 *est représentée par* rac*,* m *et* dernier*).*
2. *Ecrire la fonction de prototype* int traiteProcessus(minimier m, int dernier) *qui traite le processus à la racine de la file* f_0 *et renvoie la position du dernier élément dans le minimier rangé dans* m *en ordre direct. Cette fonction diminue de 5. la charge du processus à la racine de* f_0*, le supprime si la charge est nulle, et sinon donne à sa clé la valeur* tailleIntervalle(rac,dernier) - m[rac].priorite*, puis fait redescendre le processus dans la file en accord avec sa nouvelle clé.*
3. *On suppose écrites les fonctions* insertionProcessus_I *et* traiteProcessus_I *correspondant à la gestion des processus de la file* f_1*. Ecrire la fonction d'en-tête* void gereFileBiprocesseur(void) *qui va effectuer les actions suivantes :*
   - *Entrer une série de processus alternativement dans les deux files. Le* i^{me} *processus entré a comme pid* $100 + i$*, comme priorité* $i \bmod 3$ *et comme charge* 20 *si* i *est pair et* 70 *s'il est impair.*
   - *Tant que tous les processus ne sont pas terminés, traiter les processus à la racine des files* f_0 *et* f_1 *et éventuellement supprimer celui à la racine de* f_0 *et l'insérer dans* f_1 *(ou vice-versa) si la répartition des processus entre les deux processeurs n'est pas équilibrée.*