Eléments de correction
Examen de 1ère session - 2 juin 2010

Exercice 1 : Cours (3 points)

Pour cet exercice, notation QCM : réponse fausse = des points en moins.

Q 1.1 Tri insertion, bulle, tas, etc.

Q 1.2 Tri par casier ou par base.

Q 1.3 Un tri qui n'utilise pas d'espace mémoire supplémentaire, dit autrement un tri dont la complexité en espace est $\Theta(1)$

Q 1.4 Le noeud 2 est désiquilibré à droite mais le noeud 8 est déséquilibré à gauche : il faudra deux rotations. Rotation à droite du noeud 8 puis roatation à gauche de 2.

Exercice 2 : Structures linéaires (8 points)

Q 2.1 Une table de hachage en adressage ouvert de taille `MAX` puisqu'on pourra associer à chaque représentant d'ensembl une unique case de la table.

Q 2.2

```
1  function h (e : ENSEMBLE) : CARDINAL;
2  begin
3    h := e.tete.representant.valeur;
4  end {h};
```

Q 2.3

```
1  type TABLE_HACHAGE = record
2     valeurs : array[1..MAX] of ENSEMBLE;
3  end {record};
```

Q 2.4 L'insertion est immédiate : $\Theta(1)$, la supppression également : $\Theta(1)$. La recherche d'un ensemble à partir de son représentant est $\Theta(1)$.

Q 2.5

```
1  procedure creer_ensemble (x : CARDINAL);
2  var e : ENSEMBLE;
3  begin
4    e.tete := new CELLULE;
5    e.tete.valeur := x;
6    e.tete.suivant := NIL;
7    e.tete.representant := e.tete;
8    e.queue := e.tete;
9    inserer(t,e);
10 end {creer_ensemble};
```

Q 2.6 La création de l'ensemble est en $\Theta(1)$, l'insertion dans la table est en $\Theta(1)$. La procédure `creer_ensemble` s'effectue donc en $\Theta(1)$.

Q 2.7 Cela permet d'ajouter en queue facilement, ce qui est utile pour l'union.

Q 2.8

```
1  procedure union (x, y : CARDINAL);
2  var
3    ex, ey : ENSEMBLE;
4    p : PTRCELLULE;
5  begin
6    ex := rechercher(t,x);
7    ey := rechercher(t,y);
8    supprimer(t,x);
9    ey.queue.suivant := ex.tete;
10   ey.queue := ex.queue;
11   p := ex.tete;
12   while p <> NIL do begin
13     p.representant := ey.tete.representant;
14   end {while};
15 end {union};
```

Q 2.9 $\Theta(1)$ car il n'y a pas de comparaison.

Q 2.10 Il y a $n \times 1$ création de cellules avec les appels à `creer_ensemble`. Les appels à `union(i,i+1)` demandent i mises à jour. Soit un nombre total de création/mises à jour :

$$n + 1 + 2 + \ldots + n - 1 = \frac{n^2 + n}{2}$$

Q 2.11 Il y a $2n - 1$ opérations, soit

$$\frac{n^2 + n}{2(2n - 1)}$$

Q 2.12 `union(2,1), union(3,1), ...`

Exercice 3 : Structures arborescentes (6 points)

Q 3.1 Au minimum, et dans le pire des cas, c'est-à-dire lorsque les deix ABR sont équivalents, il sera nécessaire de comparer toutes les valeurs au moins une fois si les listes obtenues sont triées, ce qui est facile à obtenir avec des ABR, soit n comparaisons et un espace en $\mathcal{O}(n)$.

Q 3.2

```
1  function equivalents (a1,a2 : ABR) : BOOLEAN;
2  var
3    l1, l2, p, q : LISTE;
4  begin
5    l1 := LISTE_VIDE;
6    l2 := LISTE_VIDE;
7    abr2liste(a1,l1);
8    abr2liste(a2,l2);
9    p := tete(l1);
10   q := tete(l2);
11   while (p <> NIL) and (q <> NIL) and (valeur(p) = valeur(q)) do begin
```

```
12      p := reste(p);
13      q := reste(q);
14    end {while};
15    equivalents := (p = NIL) and (q = NIL);
16  end {equivalents};
17
18  procedure abr2liste(a : ABR; var l : LISTE);
19  begin
20    if a <> NIL then begin
21      abr2liste(fils_droit(a),l);
22      l := ajouter_en_tete(l,valeur(a));
23      abr2liste(fils_gauche(a),l);
24    end {if};
25  e,d {abr2liste};
```

Q 3.3 Cette procédure affiche les valeurs de l'ABR `t` dans l'ordre croissant.

Q 3.4 Le nombre d'opérations empiler et dépiler est le même : il y a aurtant d'opérations qu'il y a de noeuds dans l'arbre. La hauteur atteinte par la pile correspond à la hauteur de l'arbre.

Q 3.5 Il suffit de prendre deux piles et de mimer ce que fait `mystere` sauf que, lorsqu'on dépile, on vérifie que les valeurs trouvées sont les mêmes.

Exercice 4 : Dictionnaire (3 points)

Q 4.1

```
1  type LISTE_DE_COUPLE = record
2    lettre : CHAR;
3    fils : ^NOEUD;
4    frere : ^LISTE_DE_COUPLE;
5  end {record};
```