Institut Galilée — Algorithmique, arbres et graphes
Année 2007-2008 — L2

Partiel du 19 mai 2008

Aucun document autorisé. Le barême est uniquement indicatif.

Vous pouvez écrire les algorithmes en C ou en pseudo-code. Si cela vous pose trop de difficultés n'hésitez pas à répondre en décrivant un algorithme par des phrases ! Vous pouvez faire appel à des fonctions auxilliaires vues en cours (comparaisons, sous-tableau, etc.).

Première partie — 10 points

Exercice 1.
Rappeler les définitions utilisées et justifier (démontrer) vos réponses.

1. Est-ce que $n^2 - 2n + 1 = O(n^2)$? — 1 pt, 9 min
2. Est-ce que $\sum_{i=1}^{n} \log i = \Omega(n)$? — 1 pt, 9 min
3. Est-il vrai que si $f = O(g)$ et $f = \Omega(h)$ alors $g = \Omega(h)$? — 1 pt, 9 min

Exercice 2 (Invariant de boucle)**.**
Étant donné un tableau T de $N > 0$ entiers, l'algorithme Maximum renvoie l'indice de l'élément maximum de T. Démontrer le à l'aide d'un invariant de boucle. — 1.5 pt, 13 min

Fonction Maximum(T)
```
m = 0;
pour i ← 1 à Taille(T) − 1 faire
    si T[i] ≥ T[m] alors
        m = i;
retourner m;
```

Exercice 3 (Piles, files)**.**
Partant d'une pile vide, on ajoute (empile) 10, puis 20, 30, 40, 50 combien doit-on retirer (dépiler) d'éléments pour que le prochain élément qui sera retiré soit 40 ? (Facile) Même question en utilisant une file (initialement vide). — 0,5 pt, 4 min

Exercice 4 (Maximums successifs)**.**
Monsieur Williams écrit une application manipulant des éléments deux à deux comparables, en cherchant à en optimiser le temps d'exécution en pire cas. Une partie de l'application accède à un tableau d'éléments, désordonnés, et doit trouver un élément maximum, l'afficher et le retirer du tableau. Il peut utiliser la fonction Maximum précédente pour trouver l'indice du maximum, puis retirer cet élément (par exemple, en l'échangeant avec le dernier élément du tableau et en diminuant la taille du tableau de 1). Toute cette opération est en $\Theta(N)$ où N et la taille du tableau. Mais monsieur Williams remarque que son application doit ensuite fréquement recommencer l'opération sur ce tableau et qu'il pourrait sans doute faire mieux que dépenser des temps $\Theta(N)$ puis $\Theta(N-1)$, $\Theta(N-2)$ etc.

Il pense tout d'abord qu'il serait préférable de trier le tableau dès le départ pour que l'opération de retrait du maximum soit ensuite en $O(1)$. Mais il estime que le temps asymptotique d'exécution d'un tri par comparaison, en pire cas, est trop élevé même en choisissant le meilleur algorithme de tri.

1. Quel est ce temps ? Quel algorithme de tri le réalise ? — 1 pt, 9 min

Il trouve alors une structure de donnée qui lui permet d'arranger le tableau, en place, en un temps $\Theta(N)$ pour qu'ensuite chaque retrait du maximum se fasse assez rapidement.

2. Quelle est cette structure de donnée ? Combien de temps prennent chaque retrait du maximum, en pire cas ? — 1 pt, 9 min

3. Si T est le tableau $0, 5, 8, 3, 2, 2, 1, 9, 7$ comment est T, une fois réarrangé ? Et après un premier retrait du maximum ? Et après un second retrait du maximum ? 1 pt 9 min

Exercice 5 (Insertion / suppression ABR)**.**
Former un arbre binaire de recherche en insérant successivement et dans cet ordre les éléments : $5, 7, 3, 9, 2, 0, 6, 4, 8, 1$ (répondre en représentant l'arbre obtenu). Supprimer l'élément 5. (répondre en représentant le nouvel arbre. Il y a deux réponses correctes possibles, selon la variante choisie pour l'algorithme de suppression). 1 pt 9 min

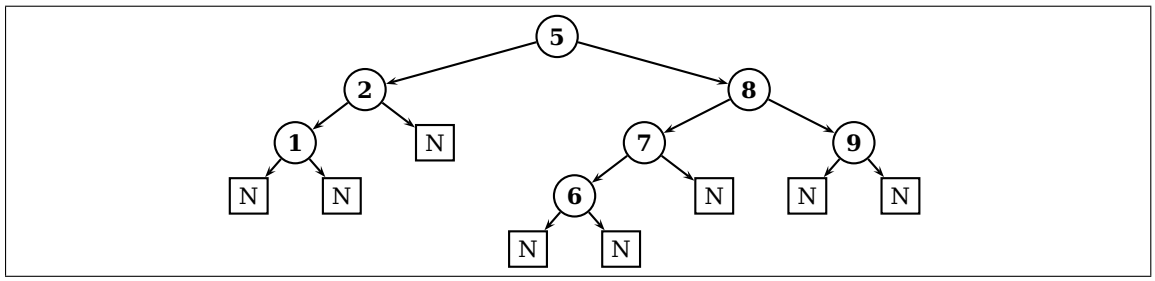

Fig. 1: Coloriage

Exercice 6.
Est-il possible de colorier tous les nœuds de l'arbre binaire de recherche de la figure 1 pour en faire un arbre rouge noir ? 1 pt 9 min

Seconde partie : problèmes 10 points

Exercice 7 (Plus grand sous-tableau à somme nulle)**.**
Soit un tableau T de N entiers. On cherche un sous-tableau de T, tel que la somme des éléments de ce sous-tableau soit nulle. De plus on souhaite que ce sous-tableau soit le plus grand possible. Un sous-tableau T' non-vide de T est donné par un couple d'indices (i, j) avec $0 \leq i \leq j \leq N-1$, les éléments de T' sont alors $T[i], T[i+1], \ldots, T[j]$.

On peut pour cela calculer toutes les sommes pour tous les sous-tableaux (non-vides) possibles de T et parmi ceux pour lesquels cette somme est nulle en trouver un de longueur maximum.

1. Quel serait le temps d'exécution d'un algorithme fondé sur cette méthode, en notation asymptotique et en fonction de N ? (Il faut expliquer comment vous écririez l'algorithme mais sans nécessairement le détailler). 1 pt 9 min

Supposons maintenant que l'on fabrique un autre tableau S de même taille que T de la manière suivante. Chaque élément $S[i]$ est un couple d'entiers. Au départ, le premier de ces deux entiers, `S[i].somme` en C, est égal à la somme $\Sigma_{j=0}^{j=i} T[j]$. Le second, `S[i].indice`, a simplement pour valeur l'indice i. On tri ensuite S par sommes croissantes à l'aide d'un tri stable.

2. Comment peut on résoudre le problème initial à l'aide de S ? (Décrire l'algorithme sans nécessairement le détailler, en supposant que S trié est fourni). 1 pt 9 min
3. Quel temps d'exécution global peut on obtenir par cette méthode (donner un temps d'exécution pour chaque étape). 1 pt 9 min

Exercice 8 (Arbre Splay)**.**
On dispose d'un ensemble de N éléments (sans répétitions) dans lequel on va réguliérement chercher un élément par sa clé (les clés sont deux à deux comparables). On décide de former un arbre binaire de recherche avec les éléments.

1. Combien de temps prendra la recherche d'un élément ? En meilleur cas ? En pire cas ? Pourquoi serait-il intéressant d'utiliser une arbre rouge noir ? (Synthétiser les différents résultats du cours). 1 pt 9 min

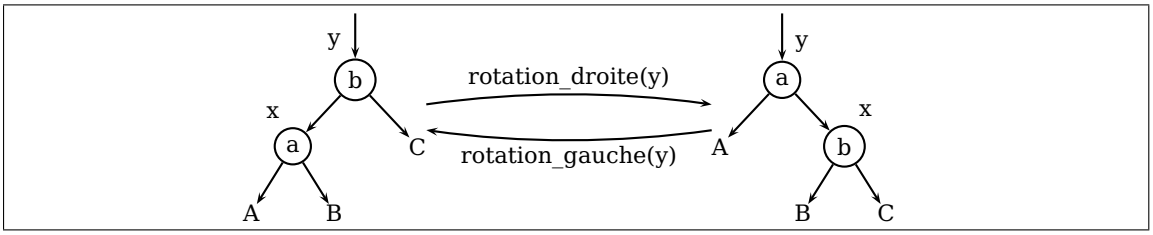

Fig. 2: Rotations gauche et droite. Dans cette version les éléments a et b sont échangés entre les nœuds x et y mais y garde sa place dans l'arbre.

En fait, certains éléments vont être recherchés plus souvent que d'autres (mais on ne sait pas lesquels) et certaines recherches seront répétées à intervalles rapprochés. Il est alors intéressant, après avoir cherché un élément, de le mettre à la racine de l'arbre. On utilise pour cela des rotations, rappelées succintement dans la figure 2 (attention comme pour l'implantation en C, le nœud centre de la rotation garde sa place dans l'arbre). La fonction Remonter(x) fait remonter à la racine l'élément a contenu dans le nœud x.

Fonction Remonter(x)

```
y = Parent(x);
si y ≠ NULL alors
    /* y existe, x n'est pas la racine                          */
    si x == Gauche(y) alors
        /* x est fils gauche de y                               */
        RotationDroite (y);
    sinon
        /* x est fils droit de y                                */
        RotationGauche (y);
    /* L'élément qui était dans le nœud x est désormais dans le nœud y   */
    Remonter (y);
```

Supposons qu'on modifie l'algorithme de recherche d'un élément de manière à ce qu'après avoir trouvé le nœud x contenant l'élément recherché, lorsqu'il existe, on fasse appel à Remonter(x).

2. Juste après la recherche d'un élément présent dans l'arbre, combien de temps prendrait une nouvelle recherche de cet élément ? Et si on cherche un élément e, puis un élément e' différent, puis à nouveau l'élément e, que e et e' sont dans l'arbre, combien de temps prendra la dernière recherche (où se trouve e après la recherche de e') ? 1 pt 9 min
3. Soit l'ABR de la figure 3, quel ABR obtient-on après avoir cherché 1 ? En repartant de l'ABR de la figure 3 quel ABR obtient-on après avoir cherché 4 ? 1 pt 9 min

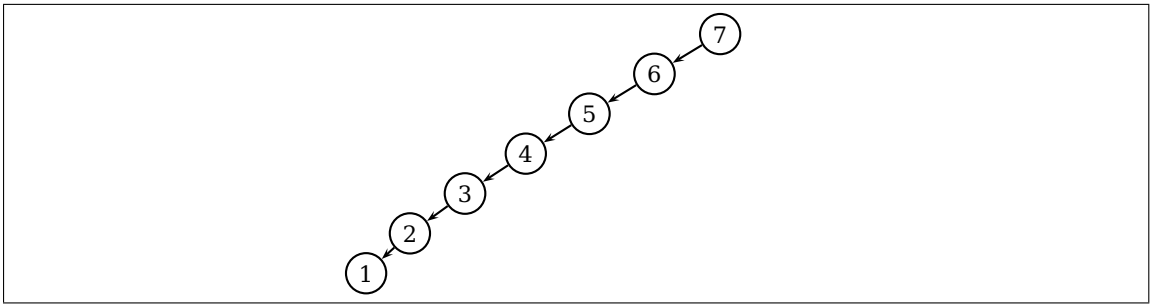

Fig. 3: Un ABR peigne.

Daniel Sleator et Robert Tarjan ont montré en 1985 qu'il était plus intéressant de faire remonter l'élément a trouvé vers la racine à l'aide d'une opération qu'ils ont appelé Splay. Cette opération est définie par cas.

– Si a est à la racine il n'y a plus rien à faire.
– Si le parent du nœud x contenant a est la racine il suffit de faire une rotation qui remonte a à la racine.

Dans les autres cas x a un parent d'élément p et un grand-parent d'élément g. Il y a deux possibilités.

– Si a est du même coté de p que p par rapport à g alors on applique une transformation *Zig-Zig* (figure 4 et cas symétrique).
– Sinon les côtés différent et on applique une transformation *Zig-Zag* (figure 5 et cas symétrique).

Après l'une de ces deux transformations il faut recommencer à appliquer Splay à partir la nouvelle position de a dans l'arbre.

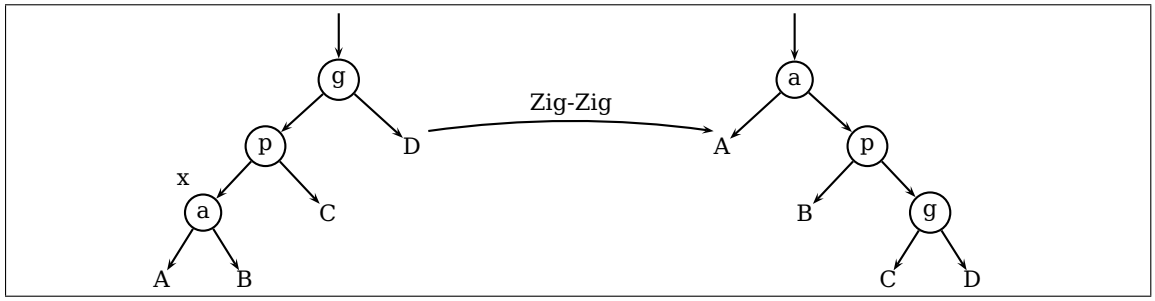

Fig. 4: Étape Zig-Zig

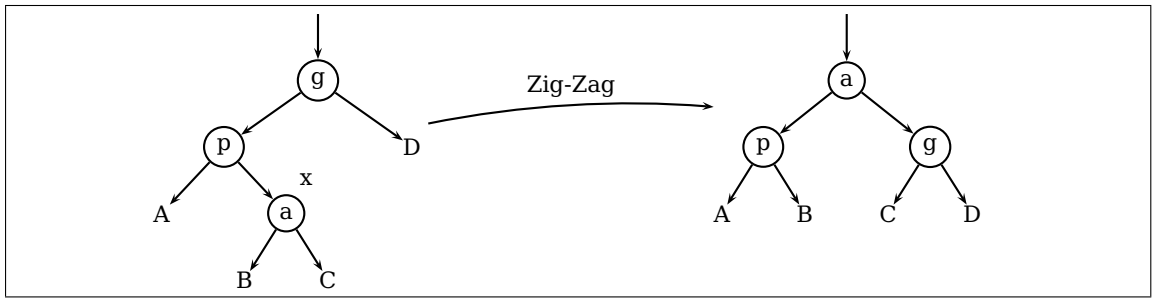

Fig. 5: Étape Zig-Zag

4. Exprimer les transformations Zig-Zig et Zig-Zag à l'aide des rotations. Écrire une fonction pour chacune de ces transformations. 1.5 pt 13 min
5. Quelle étape de l'algorithme Splay le rend différent de Remonter ? 1 pt 9 min
6. Écrire l'algorithme Splay. 1.5 pt 13 min

200