Université de Mons
FS/1PREPAA.P.MAS.IN/6584 – Algorithmique
Examen de seconde session – Partie théorique

Le 23 août 2010

Consignes

- Pour cette partie, vous n'avez pas le droit d'utiliser de notes.
- Cette partie de l'examen dure 1 heure 15 minutes.
- Veillez à bien justifier vos réponses. Une réponse mal justifiée, même correcte, ne permet pas d'obtenir le maximum des points.
- Quand vous indiquez une complexité, veillez à bien expliquer ce que sont les paramètres qui apparaissent dans le $\mathcal{O}$. Par exemple, $\mathcal{O}(n^2)$ n'a aucun sens si n n'apparaît pas dans l'algorithme ou dans la définition de la structure qui est traitée…

Question 1 – 4 points

Expliquez ce qu'est une *liste circulaire avec élément pré-tête*, et en quoi ce type de liste diffère des *listes simplement liées*. Donnez un algorithme qui est amélioré par l'utilisation de l'élément pré-tête, expliquez en quoi l'élément pré-tête est utile, et dans quelle mesure il permet d'améliorer les performances de l'algorithme.

Correction

Voir syllabus pp. 60 *sqq*.
Exemple de réponse-type :

1. Une liste circulaire est une liste dont le pointeur *suivant* du dernier élément ne contient pas la valeur **NIL** (comme dans une liste simplement liée), mais pointe vers le premier élément de la liste. Autrement dit, le dernier élément de la liste est caractérisé par le fait que son champ *suivant* contient la même adresse que la tête.
2. Une liste avec *élément pré-tête* est une liste dont le premier élément n'est pas significatif : la valeur qu'il contient ne fait pas *logiquement* partie de la liste. Ainsi, une liste *vide* (logiquement) contient toujours un élément, à savoir l'élément *pré-tête*. Au contraire, dans une liste simplement liée classique, la première information représentée par la liste est celle qui est stockée dans le premier élément de la liste (celui pointé par la tête).

Une liste *circulaire avec élément pré-tête* est une liste qui possède ces deux caractéristiques. *Remarque* : il fallait bien expliquer les *deux termes*.
Un exemple d'algorithme qui est amélioré par la présence de l'élémént pré-tête et le fait que la liste est circulaire est l'algorithme de *recherche*. Le principe consiste à stocker la valeur recherchée dans l'élémént pré-tête, et à commencer la recherche sur le second élément physique (donc le premier élément logique, celui qui suit l'élément pré-tête), et d'avoir une boucle de parcours dont le test est uniquement $p.\text{info} \neq k$, ù k est la valeur recherchée. Cette boucle s'arrêtera toujours puisque l'élément pré-tête sera, au pire cas (c'est-à-dire au cas où la valeur k n'est pas dans la liste), atteint, par le fait que la liste est circulaire. Cette version de l'algorithme est plus efficace, car on ne doit faire qu'un seul test par élément (au lieu de 2 dans le cas « classique » : $p \neq \textbf{NIL}\ p.\text{info} \neq k$). Cela n'améliore donc pas la complexité en terme de $\mathcal{O}$, mais diminue tout de même la constante multiplicative. *Remarque* : il importait d'expliquer 1) l'algorithme lui-même (éventuellement en le donnant), 2) en quoi l'algorithme exploite le caractère circulaire avec pré-tête de la liste, 3) quelle est la portée de l'amélioration.

Question 2 – 6 points

Donnez l'algorithme de recherche dichotomique (version récursive). Détaillez-en le fonctionnement étape par étape sur le vecteur ci-dessous, pour recherche la valeur 45 :

2	6	12	15	27	32	45	76	77	78	81	89	95	101	125

En théorie, quel est la profondeur maximale des appels récursifs effectués par cet algorithme (sur un vecteur de taille n) ? Justifiez brièvement (pas de démonstration). L'exemple ci-dessus confirme-t-il la théorie ? Justifiez.

Correction

Voici les étapes d'exécution :

bi							*m*							*bs*
2	6	12	15	27	32	45	76	77	78	81	89	95	101	125

bi			*m*			*bs*								
2	6	12	15	27	32	45	76	77	78	81	89	95	101	125

				bi	*m*	*bs*								
2	6	12	15	27	32	45	76	77	78	81	89	95	101	125

						bi, *bs*								
2	6	12	15	27	32	45	76	77	78	81	89	95	101	125

En théorie, la profondeur maximale des appels récursifs, sur un vecteur de taille n, sera de $\log_2(n)+1$, au maximum. Intuitivement, cela provient du fait que l'algorithme divise par deux la taille de l'intervalle de recherche à chaque appel récursif. Donc, pour augmenter le nombre total d'appels récursifs de 1, il faut multiplier la taille du vecteur par 2. Le nombre d'appels récursifs est donc proportionnel à $\log_2(n)$. Le $+1$ sert à compter l'appel initial. Dans le cas présent $\log_2(n)+1 = 3,91+1 = 4,91$. Effectivement, la profondeur atteinte dans les appels récursifs est de 4. *Remarque :* dire que l'algorithme est en $\mathcal{O}(\log(n))$ ne répond pas à la question car cela ne caractérise pas précisément le *nombre* d'appels récursifs. En effet, le $+1$ est nécessaire dans le cas présent.
