ÉCOLE POLYTECHNIQUE DE MONTRÉAL
Département de génie informatique
Cours INF4705 : Analyse et conception d'algorithmes (Automne 2004)
3 crédits (3-1.5-4.5)

EXAMEN FINAL
CORRIGÉ

DATE : Mardi, le 14 décembre 2004

HEURE : 9H30 à 12H00

DURÉE : 2H30

NOTE : Aucune documentation permise. Aucune calculatrice permise.

Ce questionnaire comprend 6 questions sur 3 pages pour 20 points.

Bon travail et joyeuses fêtes!

Question 1 : Vrai ou Faux ? [3 points]

a) **[1 point]** Un algorithme de complexité O(nlogn) dans le cas moyen est toujours plus rapide qu'un algorithme dont la complexité est $O(n^2)$ dans le cas moyen.

Faux.

b) **[1 point]** Pour tout graphe connexe, il existe un et un seul arbre de recouvrement minimum.

Faux.

c) **[1 point]** Les algorithmes de parcours de graphes branch-and-bound trouvent toujours la solution optimale.

Vrai.

Question 2 : Pot-pourri [6 points]

a) **[1 point]** Expliquez la différence entre les notations O() et Ω(), sans donner leur définition formelle.

O() exprime une borne supérieure sur la consommation de ressources. Ω() exprime une borne inférieure sur la consommation de ressources.

b) **[1 point]** Les algorithmes diviser-pour-régner et de programmation dynamique sont tous deux basés sur une relation de récurrence. Expliquez cependant la différence fondamentale entre les deux techniques de conception.

Diviser-pour-régner procède de haut en bas, alors que la programmation dynamique procède de bas en haut.

c) **[1 point]** Qu'est-ce qu'un algorithme vorace ?

Un algorithme vorace est un algorithme qui fait toujours le choix qui lui semble le meilleur sur le moment. Autrement dit, il fait le choix localement optimal, dans l'espoir que ce choix conduira à une solution globalement optimale.

d) **[1point]** Expliquez brièvement le fonctionnement général des heuristiques d'amélioration locale. Quelle est la lacune la plus évidente de ces heuristiques ?

À partir d'une solution initiale s_0 , on applique une succession de modifications locales afin d'obtenir une meilleure solution s_1. On peut répéter cette opération pour obtenir une solution encore meilleure s_2, etc… On arrête dès qu'une solution s_i n'est pas meilleure que la solution s_{i-1}. La principale lacune des ces heuristiques est qu'elles sont susceptibles d'être « piégées » dans un optimum local voisin de la solution initiale.

e) **[1 point]** Expliquez comment vous transformeriez un algorithme Monte Carlo en un algorithme Las Vegas.

Soit A un algorithme Monte Carlo, et soit B un algorithme qui vérifie la correction d'une solution de A. Soit C un algorithme qui exécute A en boucle, jusqu'à ce que la solution retournée soit reconnue correcte par B. L'algorithme C est un algorithme Las Vegas.

f) **[1 point]** Les algorithmes de programmation dynamique reposent sur le principe d'optimalité. Rappelez la définition du principe d'optimalité.

Une solution optimale d'un exemplaire quelconque d'un problème d'optimisation peut être construite à partir des solutions optimales de sous-exemplaires de ce problème.

Question 3 : Algorithmes voraces [3 points]

Vous désirez vous rendre en automobile de Montréal à Vancouver. Le réservoir de votre voiture vous permet de parcourir une distance D lorsqu'il est plein. Au cours du voyage, vous rencontrerez *k* postes d'essence où vous pourrez faire le plein.

Soit :

- *di* la distance pour aller du poste *i*–1 au poste *i*
- *d*1 la distance de Montréal au poste 1
- *dVancouver* la distance du poste *k à Vancouver*
- *C* = { *c1*, *c2*, ..., *ck*} l'ensemble des postes d'essence.

On suppose $d1 \leq D$, $di \leq D$, $d\text{Vancouver} \leq D$.

On désire faire le moins d'arrêts possibles.

a) **[2 points]** Écrivez un algorithme vorace qui détermine les postes où vous devriez vous arrêter.

```
FONCTION Vorace (C)
    S = Ø    // À Montréal, aucun poste n'a été choisi
    TANT que Vancouver n'est pas atteint
        i = prochain poste rencontré
        Si le plein est nécessaire pour se rendre au poste i+1
            Ajouter i à S      // faire le plein
    Retourner S
```

b) **[1 point]** Votre voiture a une autonomie de 600 km (D = 600 km), et vous rencontrez 10 postes d'essence C1 à C10. En appliquant l'algorithme que vous avez écrit, et en utilisant le tableau des distances ci-dessous, indiquez les postes d'essence où vous vous arrêterez, sachant que le nombre de ces postes d'essence doit être minimal.

Tableau des distances entre les postes d'essence		
Point ou poste d'essence de départ	Point ou poste d'essence d'arrivée	Distance (km)
A	C1	550
C1	C2	100
C2	C3	225
C3	C4	300
C4	C5	100
C5	C6	50
C6	C7	75
C7	C8	100
C8	C9	300
C9	C10	175
C10	B	400

Solution : C1, C3, C7, C10

Question 4 : Algorithmes branch-and-bound [3 points]

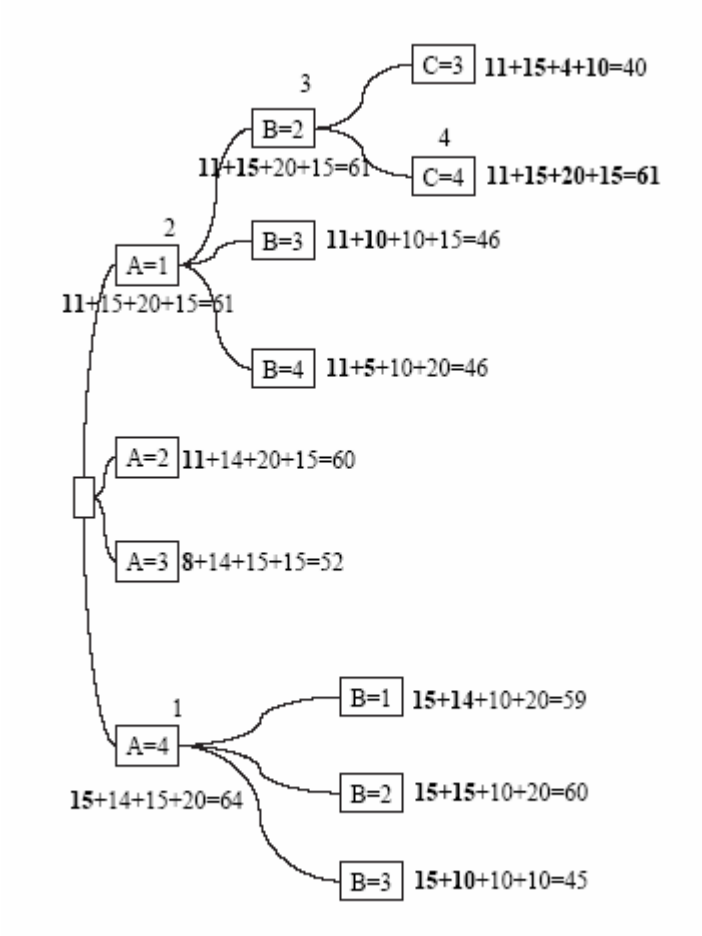
a) **[1 point]** Les algorithmes branch-and-bound s'articulent autour de 3 composantes essentielles. Quelles sont ces composantes ?

1- Une fonction fournissant une borne inférieure (ou supérieure, selon le cas) pour la meilleure solution réalisable le long du sous-arbre en cours.
2- Une stratégie de sélection des sous-arbres à élaguer et du prochain sous-arbre à explorer en priorité.
3- Une règle de branchement à appliquer à un sous-arbre qui vient d'être découvert, et qui ne peut être ni élagué, ni exploré en priorité.

b) **[2 points]** Vous avez quatre objets à vendre (*A*, *B*, *C*, *D*) et quatre acheteurs (1, 2, 3, 4). Chaque acheteur est prêt à payer un certain prix pour chaque objet comme indiqué dans le tableau ci-dessous (par exemple, l'acheteur 1 est prêt à payer 11$ pour l'objet *A*). Utilisez la méthode branch-and-bound pour trouver à quel acheteur vous allez vendre chaque objet afin de maximiser votre profit sachant qu'un acheteur achète un objet et un seul. Vous devez expliciter la partie du graphe qui sera explorée (avec le calcul des bornes) en plus de fournir la solution. De plus, indiquez clairement et complètement l'ordre de parcours du graphe au cours de l'exécution de l'algorithme.

	1	**2**	**3**	**4**
A	11	11	8	15
B	14	15	10	5
C	3	3	4	15
D	10	10	20	10

Solution : a=1, b=2, c=4, d=3; 11+15+20+15=61



Question 5 : Algorithmes de programmation dynamique [3 points]

Vous participez à une course à bicyclette. Le long du parcours, vous retrouvez *N* points de ravitaillement (incluant les points de départ et d'arrivée). À chaque point de ravitaillement, vous pouvez louer une bicyclette que vous pouvez rendre à n'importe quel point de ravitaillement

suivant du parcours. Or, il peut arriver qu'une location de *i* à *j* coûte plus cher qu'une série de locations plus courtes. Dans un tel cas, vous avez intérêt à rendre la bicyclette à un point de ravitaillement *k* entre *i* et *j*, et à louer une autre bicyclette pour continuer la course.

a) **[2 points]** En adoptant une approche de programmation dynamique, donnez une expression du coût minimal pour effectuer le trajet au complet (c'est-à-dire en franchissant les *N* points de ravitaillement).

Soient C[i, j] le coût de location entre i et j (i < j), et D[k] le coût minimal pour effectuer le trajet de 1 à k (k = 2, 3, ..., N).

Alors :

$$D[N] = Min\ (C[1,N],\ \min_{j=2}^{N-1} (D[j] + C[j,N]))$$

b) **[1 point]** Quel est le temps de calcul nécessaire pour évaluer ce coût minimal en fonction de *N* ?

Le calcul de chaque D[k] est linéaire en k. Donc, le temps de calcul est en $O(N^2)$.

Question 6 : Algorithmes probabilistes [2 points]

a) **[1 point]** Vous disposez d'un algorithme Monte Carlo MC_1 (51/100)-correct et vrai-biaisé pour un problème de décision (c'est-à-dire un problème dont la solution est **Vrai** ou **Faux**). Expliquez ce que signifient les expressions « algorithme Monte Carlo », « (51/100)-correct » et « vrai-biaisé ».

algorithme Monte Carlo : algorithme probabiliste donnant une mauvaise réponse à l'occasion, mais ayant une probabilité élevée de retourner la bonne réponse, indépendamment de l'exemplaire considéré.

(51/100)-correct : la probabilité que l'algorithme retourne la bonne réponse est au moins égale à (51/100), quel que soit l'exemplaire considéré.

vrai-biaisé : l'algorithme est toujours exact quand la réponse est ***Vrai****.*

b) **[1 point]** Soit $5MC_1$ un algorithme qui exécute au plus 5 fois l'algorithme MC_1 précédent, et qui retourne **Faux** si et seulement si MC_1 a retourné 5 fois **Faux**. L'algorithme $5MC_1$ retourne **Vrai** dès que l'algorithme MC_1 retourne **Vrai**. Calculez la probabilité d'échec de l'algorithme $5MC_1$.

p_echec $= Prob(5MC_1$ *retourne* ***Faux*** *lorsque la bonne réponse est* ***Vrai****)*
$= Prob(MC_1$ *retourne 5 fois* ***Faux*** *lorsque la bonne réponse est* ***Vrai****)*
$= (1 - (51/100))^5$
$= 0.028$