Questionnaire, 20 points

Répondre en indiquant sur une feuille séparée, pour chaque numéro de question **V** (vrai) ou **F** (faux). Les réponses incorrectes sont pénalisées.

1 Pour toute fonction $f(n)$ sur $\mathbb{N}$ asymptotiquement positive ou nulle, on a : $f(n) + o(f(n)) \subseteq O(f(n))$ Vrai

2 Toute solution de la récurrence $T(n) = 3T(n/3) + O(\sqrt{n})$ appartient à $\Omega(n \log(n))$. Faux

Des fonctions linéaires sont solutions de la récurrence.

3 Dans le pire des cas, le temps d'exécution de *Quicksort* sur un tableau de taille n est $\Omega(n^2)$. Vrai

Par exemple si le pivot choisi est toujours le premier élément du sous-tableau courant et si le tableau est déjà trié.

4 Trier par comparaisons un tableau de n valeurs prises dans $0 \ldots 9$ requiert $\Omega(n \log n)$ comparaisons. Faux

Ce problème est une généralisation du problème du drapeau hollandais. On peut le résoudre en temps linéaire, en appliquant 10 procédures de pivot. L'argument de comptage classique donne une borne inférieure en $\omega(n)$: il n'y a que 10^n applications de $\{1, \ldots n\}$ dans $\{0 \ldots 9\}$.

5 Dans un graphe, l'arbre de recouvrement de poids minimal est unique lorsque toutes les arêtes portent des poids différents. Vrai

Remarquer qu'à chaque étape des algorithmes classiques (Prim-Dijkstra, Kruskal), il n'y pas d'alternative.

6 Un arbre binaire de recherche sur n entiers peut être construit en temps $O(n)$. Faux

Sinon on saurait trier en utilisant un nombre linéaire de comparaisons.

7 Dans une version probabiliste de *Quicksort*, on applique au tableau fourni en entrée une permutation aléatoire (tirée uniformément dans l'ensemble des permutations sur n éléments). Dans le pire des cas, l'espérance du temps de calcul de *Quicksort* probabiliste est $\Omega(n^2)$. Faux

Quelque soit le tableau de départ, aprés avoir appliqué une permutation aléatoire, le tableau est trié aléatoirement (on profite du fait que les permutations forment un groupe). L'espérance du temps de calcul de *Quicksort* est alors donnée par l'analyse en moyenne.

8 Dans un graphe orienté arbitraire, on peut trouver un tri topologique en temps linéaire. Faux

Certains graphes orientés n'admettent pas de tri topologique.

9 On dispose de trois listes triées à trois éléments, on veut trouver le $3^{\text{ème}}$ plus petit élément dans l'ensemble constitué par la réunion de ces trois listes. On peut faire cela en 3 comparaisons. Faux

L'élément recherché peut être n'importe lequel des neufs éléments. Si trois comparaisons étaient suffisantes, l'arbre de décision décrivant le comportement de l'algorithme aurait au plus huit feuilles. Une des positions serait donc exclue a priori, ce qui est incorrect.

10 En réduisant le problème du tri à la construction d'un tas (*heap*), on peut prouver que construire un tas sur n éléments requiert $\Omega(n \log n)$ comparaisons. Faux

Les tas sont constructibles en temps linéaire. Dans heapsort, c'est la phase d'exploitation du tas qui coûte $\Omega(n \log n)$ opérations.

Toutes les questions sont indépendantes.

Problème 1, 20 points

Donner un algorithme efficace pour résoudre le problème suivant.

Donnée : Un ensemble de n paires de skis est offert à un groupe de m skieurs ($m \leq n$). L_i désigne la hauteur de la $i^{\text{ème}}$ paire de skis, et ℓ_j désigne la hauteur du $j^{\text{ème}}$ skieur. A priori, on souhaiterait que chaque skieur reçût une paire de skis de même hauteur que lui. Comme ce n'est pas toujours possible, on se contente de minimiser les écarts en répondant à la question suivante.

Question : Trouver une affectation des skis aux skieurs (i.e. une fonction injective ϕ de $\{1 \ldots m\} \to \{1 \ldots n\}$) telle que $\sum_i |L_{\phi(i)} - \ell_i|$ soit minimale.

Remarque : Chaque skieur doit recevoir une paire de skis.

Suggestions : Résoudre d'abord le problème lorsque $m = n$; puis analyser les sous-problèmes $A(k, h)$ où on considère les h plus petits skieurs et les k plus courts skis.

Corrigé

Sans perdre en généralité on supposera que les skieurs sont triés par ordre croissant. Une affectation ϕ est dite monotone si
$\forall i, j \in \{1 \ldots m\}, \quad i < j \Rightarrow L_{\phi(i)} \leq L_{\phi(j)}$.

Toute affectation optimale est monotone. En effet si ϕ est optimale sans être monotone, il existe $i, j \in \{1 \ldots m\}, \quad i < j$ et $L_{\phi(i)} > L_{\phi(j)}$. Mais alors l'affectation ϕ', telle que $\phi'(i) = \phi(j)$, $\phi'(j) = \phi(i)$ et $\phi'(k) = \phi(k)$ pour $k \notin \{i, j\}$ est de coût inférieur. En effet

$$\sum_k |L_{\phi'(k)} - \ell_k| = \sum_{k \notin \{i,j\}} |L_{\phi(k)} - \ell_k| + |L_{\phi(j)} - \ell_i| + |L_{\phi(i)} - \ell_j|.$$

Mais $|L_{\phi(j)} - \ell_i| + |L_{\phi(i)} - \ell_j| \leq |L_{\phi(i)} - \ell_i| + |L_{\phi(j)} - \ell_j|$. Il suffit d'envisager les 6 cas possibles.

Donc s'il y a autant de skis que de skieurs, l'affectation optimale est l'unique affectation monotone. On la détermine en triant les skis par taille croissante.

Cas général.

On suppose maintenant que les skis sont aussi triés par ordre croissant. Une affectation optimale ϕ possède la *propriété d'optimalité des sous-structures*: elle induit une affectation optimale des skis $\{0 \ldots \phi(m) - 1\}$ parmi les $m - 1$ plus petits skieurs.

Pour résoudre le problème général, on peut utiliser la *programmation dynamique* : on construit une table C, pour $k \geq h$ telle que $C(k,h))$ désigne le coût optimal d'une affectation des k plus petits skis parmi les h plus petits skieurs. La propriété d'optimalité des sous-structures implique la relation suivante :

$$\text{pour} \quad k+1 > h, \quad C(k+1,h) = \min(C(k,h), C(k,h-1) + |L_{k+1} - \ell_h|). \tag{1}$$

Pour calculer la première colonne de C, on utilise la règle suivante:

$$C(k+1,1) = \min(C(k,1), |L_{k+1} - \ell_1|)$$

avec $C(1,1) = |L_1 - \ell_1|$.

L'algorithme général est le suivant:

$C(1,1) = |L_1 - \ell_1|$
Pour k de 1 de à n faire $C(k+1,1) = \min(C(k,1), |L_{k+1} - \ell_1|)$
Pour h de 1 à m calculer $C(h,h)$.

Pour h de 2 de à m faire
Pour k de $h+1$ à n faire
$C(k,h) = \min(C(k-1,h), C(k-1,h-1) + |L_k - \ell_h|)$.
FinFaire
FinFaire

Pour déterminer l'affectation optimale :
h := n ;
Pour k de m à 1 faire
Si $k = h$ ou $C(k,h) < C(k-1,h)$ alors $\phi(h) := k$; $h := h-1$ FinSi ;
Si $h = 0$ alors Sortir ;
FinFaire ;

On peut donc déterminer une affectation optimale en temps $O(n \times m)$.

Problème 2, 20 points

Dans une graphe orienté sans boucle, à n sommets, une *source* est un sommet de degré entrant nul, et de degré sortant $n-1$.

Donner un algorithme qui examine $O(n)$ entrées de la matrice d'adjacence pour décider si un graphe sans boucle possède une source.
Est-il nécessaire d'examiner $\Omega(n)$ entrées?

Corrigé

Pour vérifier qu'un sommet est une source, il faut vérifier que son degré

entrant soit nul $(n-1)$ examens et vérifier que son degré sortant est $n-1$, donc il est nécessaire de faire au moins $2n-2$ examens.

Avant tout on notera qu'un graphe contient au plus une source. Si on teste une entrée $A(i,j)$, si $A(i,j)=0$ alors i n'est pas une source, sinon j n'est pas une source.

On part d'une liste de n candidats. Tant que la liste contient au moins deux candidats, on en choisit deux, appelons les i et j, et on teste $A(i,j)$. Ce qui permet d'éliminer l'un ou l'autre de la liste. En $n-1$ étapes, on a éliminé tous les candidats sauf un. On vérifie au prix de $2n-2$ examens s'il s'agit bien d'une source. On peut donc déterminer la présence d'une source en au plus $3(n-1)$ examens (en fait il suffit de $3(n-1)-\log_2 n$ examens)

Problème 3, 20 points

Dans un graphe orienté G, un *circuit* est une suite de sommets $v_0, v_1 \ldots v_k$ telle que $v_0 = v_k$ et $\forall i, 0 \leq i < k \quad (v_i, v_{i+1})$ est un arc de G. Le circuit est impair si k est impair. Donner un algorithme efficace (polynomial) pour résoudre le problème suivant:

Donnée : Un graphe orienté décrit par sa matrice d'adjacence.
Question : Le graphe contient-il un circuit impair?

Remarque: on se demandera quelle est la relation entre l'existence de chemins de longueur k entre deux points et les puissances de la matrice d'adjacence.

Corrigé

On doit remarquer qu'un circuit impair contient toujours au moins un circuit simple impair (le circuit peut se décomposer en une union disjointe de circuits simples, s'ils sont tous pairs le circuit composé l'est aussi).

On note A la matrice d'adjacence du graphe. On définit une multiplication matricielle sur la structure $(\{0,1\}, \vee, \wedge)$. $A^k[i,i] \neq 0$ si et seulement si il existe un circuit de longueur k passant par i. Pour répondre à la question, il suffit d'examiner les termes diagonaux des puissances impaires A. Il suffit par ailleurs de considérer les n premières puissances de A, puisque s'il existe des cycles impairs, il en existe de longueur inférieure à n.

Il suffit donc de calculer n puissances de A et d'en examiner les n termes diagonaux ($O(n^4)$ opérations.

Problème 4, 20 points

Un algorithme détermine le maximum dans un tableau d'éléments non triés. Pour cela il utilise une boîte noire qui effectue les comparaisons entre éléments et lui retourne le résultat. Lors de la conception de l'algorithme,

on apprend que la boîte noire peut retourner un résultat faux (mais pas plus d'une fois au cours du déroulement de l'algorithme).

Montrer que $2n-1$ appels à la boîte noire sont nécessaires et suffisants si on veut garantir que l'algorithme détermine toujours le maximum, quelque soit le tableau d'entrée et la défaillance de la boîte noire.

Corrigé
$2n-1$ appels sont suffisants: on effectue deux calculs du maximum en utilisant la boite noire comme une sous-routine entièrement fiable. Cela requiert $2(n-1)$ appels. On compare les deux résultats et on choisit le plus grand. S'il n'y a pas eu de défaillance lors des deux calculs de maximaux, le résultat est nécessairement correct. S'il y a eu une défaillance lors d'un des deux calculs de maximum, elle est compensée par la dernière comparaison.

$2n-1$ appels sont nécessaires. Pour qu'un algorithme soit correct, il faut qu'à la fin, on dispose d'une preuve de ce que les $n-1$ éléments qui n'ont pas été choisis ne sont pas le maximum. Pour être sûr qu'un élément n'est pas le maximum, il faut qu'il ait perdu au moins deux comparaisons. Cela implique donc immédiatement que $2(n-2)$ comparaisons sont nécessaires.

Pour montrer qu'il en faut $2n-1$ en pire cas, fixons une entrée quelconque, c'est à dire un ordre sur les éléments. La stratégie de la boîte noire est la suivante: elle dit toujours la vérité sauf dans la 1ère comparaison qui fait intervenir le maximum de la liste fixée (le maximum doit aussi participer aux comparaisons, il est le seul à pouvoir éliminer le 2e plus grand). Cette comparaison fait remonter le nombre total à $2n-1$.

Problème 5, 20 points

Dans un graphe non orienté, $G=(V,E)$, un *cycle* est un ensemble d'arêtes du graphe $\{(v_0,v_1),(v_1,v_2)\ldots(v_{k-1},v_0)\}$. Un cycle est simple si $|\{v_0,v_1,v_{k-1}\}|=k$. Le cycle est pair s'il contient un nombre pair d'arêtes (k est pair). L'ensemble vide désigne le cycle vide.

Deux cycles sont arête-disjoints s'ils sont disjoints en tant qu'ensembles d'arêtes. La différence symétrique de deux ensembles est formée par leur réunion moins leur intersection.

On note $\mathcal{P}(E)$ l'ensemble des ensembles d'arêtes. $\mathcal{C}$ est le sous-ensemble de $\mathcal{P}(E)$ constitué par les ensembles d'arêtes formés par des réunions de cycles disjoints (aux arêtes). On admettra que $\mathcal{C}$ muni de la différence symétrique et de la multiplication par $\{0,1\}$ est un espace vectoriel sur $\{0,1\}$ (c'est l'espace des cycles, sous-espace de $\mathcal{P}(E)$ muni des mêmes opérations).

1) Donner un algorithme polynomial en $|E|$ qui construit une base de l'espace des cycles.

2) Donner un algorithme polynomial qui détermine si un graphe non orienté contient un cycle simple pair.

Corrigé

1) La descente en profondeur permet de construire une base de l'espace des cycles. En effet si partant d'une liste vide, chaque fois que l'on rencontre un arc de retour, on ajoute à cette liste le cycle formé par le chemin d'arbre unissant les deux extrémités de l'arc arrière et l'arc arrière lui même, on construit une famille libre de cycles (on ne peut pas engendrer l'un des éléménts de la liste à partir des autres puisque chaque arc arrière appartient à un seul élément de liste).

Cette famille est aussi génératrice de l'espace des cycles. On montre que tout cycle simple peut être engendré par la famille en raisonnant par induction sur le nombre d'arcs arrières du cycle simple.

2) Si deux cycles C_1 et C_2 sont deux cycles de la base qui partagent des arètes communes, ces arètes communes forment un chemin d'arbre et alors C_1, C_2 ou $C_1 \Delta C_2$ contient un cycle pair : si on additionne les longueurs de C_1, C_2 et du cycle construit en effectuant la différence symétrique de C_1 et C_2, chaque arète de C_1 et de C_2 est comptée deux fois. La somme de trois nombres pairs ne peut être impaire.

Si tous les cycles de bases sont disjoints et impairs, le graphe ne peut contenir de cycle simple pair.

Problème 6, 10 points

La médiane d'une suite $(x_1, x_2 \ldots x_n)$ de n éléments distincts d'un ensemble totalement ordonné est un indice i $1 \leq i \leq n$ tel que

$$\#\{j \quad ; \quad x_j < x_i\} = \lfloor n/2 \rfloor \qquad \text{et} \qquad \#\{j \quad ; \quad x_j \geq x_i\} = \lceil n/2 \rceil$$

Quel est l'ordre de grandeur du nombre de comparaisons realisées par un algorithme optimal pour trouver la médiane de deux listes triées?
Vous devez donner un algorithme calculant cette médiane et réalisant un nombre de comparaisons de cet ordre de grandeur et un argument qui explique pourquoi cet ordre de grandeur est optimal.

Corrigé

La médiane peut se trouver à n'importe laquelle des $2n$ positions. Considèrons l'arbre de décision binaire qui décrit le comportement d'un algorithme de détermination de la médiane. A chaque comparaison correspond un nœud, selon le résultat on suit la branche gauche ou la droite. Cet arbre doit comporter au moins $2n$ feuilles. Il doit donc être de profondeur au moins $\log 2n$. Dans le pire des cas, il faut donc au moins $\log 2n$ comparaisons.

On peut atteindre cette borne grâce à l'algorithme récursif suivant:

Si les deux listes sont de taille 1, on détermine la médiane en une comparaison. Si les deux listes sont de longueur $n > 1$, on compare les éléments en position $\lceil n/2 \rceil$ des deux listes. Si l'élément provenant de la première liste est le plus petit des deux, aucun des éléments qui le précèdent dans la première liste ne peut être la médianne, et symétriquement aucun des éléments de position strictement supérieure à $\lceil n/2 \rceil$ ne peut être candidat. En revanche la médiane des deux sous-listes de taille $\lceil n/2 \rceil$ formées par les plus grands éléments de la première liste et les plus petits éléments de la seconde liste est aussi la médiane de l'ensemble de départ.

Le temps d'exécution de cet algorithme est solution de la récurrence :

$$T(n) = 1 + T(\lceil n/2 \rceil) = k + T(\lceil n/2^k \rceil).$$

Il appartient à $O(\log n)$.