Université Pierre et Marie Curie **Master Informatique**
23 Janvier 2006 **MI015-ALGAV**

Examen

Durée : 2h. Tous documents autorisés.
Le barème donné (sur 30) est indicatif.
La note maximale de 20/20 est attribuable à une copie qui traiterait les 2/3 de l'énoncé.
Vous pouvez donc choisir les questions que vous traitez.

1 Arbre couvrant minimum dans le plan [10 points]

On considère le problème de déterminer l'arbre couvrant minimal d'un ensemble de points du plan, lorsque la distance entre 2 points est la distance euclidienne (longueur du segment). On suppose que les distances entre tous les couples de points sont distinctes et que tous les points ont des abscisses différentes.
On va tenter successivement plusieurs algorithmes pour résoudre ce problème : les deux premiers sont incorrects (et on demande de trouver un contrexemple) et le troisième est correct (il s'agira donc de le prouver et de l'implanter)

1- Premier algorithme : montrer que cet algorithme renvoie un arbre couvrant mais qui n'est pas forcément minimum

`Trier les points par ordre d'abscisses croissante pour obtenir la suite` $P_1, ..., P_n$`,`
`Pour chaque` $i = 2, ..., n$`, connecter` P_i `avec son plus proche voisin parmi` $P_1, ..., P_{i-1}$`.`

2- Deuxième algorithme : montrer que cet algorithme renvoie un arbre couvrant mais qui n'est pas forcément minimum

```
Séparer l'ensemble des points en deux parties égales (à 1 près),de part et d'autre d'une droite
Construire récursivement un arbre couvrant minimal pour chacune des deux parties,
Connecter ces 2 arbres entre eux à coût minimal.
```

3- Troisième algorithme : cet algorithme construit un arbre couvrant minimum

```
Au départ chaque point est un arbre isolé (ensemble vide d'arcs couvrants)
Traiter les segments par ordre de longueur décroissante :
- Si le segment connecte deux arbres différents, le rajouter à l'ensemble des arcs
couvrants, pour former un nouvel arbre,
- Si le segment connecte deux points x et y du même arbre, le rajouter à l'ensemble des arcs
couvrants et supprimer le plus long segment sur le chemin entre x et y.
```

3-a) Donner un exemple d'exécution de cet algorithme.

3-b) Montrer que l'algorithme construit un arbre couvrant et que cet arbre est de coût minimum.

3-c) Étudier la complexité de l'algorithme en fonction du nombre n de points.

3-d) Décrire une implantation efficace de l'algorithme.

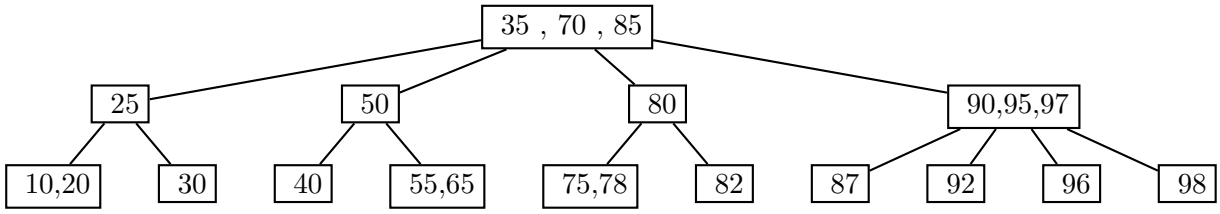
2 Concaténation et Coupure d'arbres 2-3-4 [10 points]

L'opération de *concaténation* prend en entrée deux arbres 2-3-4, T_1 et T_2, et un élément x, tels que toutes les étiquettes (éléments) de T_1 sont strictement inférieures à x et x est strictement inférieur à toutes les étiquettes de T_2. Et la *concaténation* renvoie comme résultat l'arbre 2-3-4 contenant x et tous les éléments de T_1 et T_2.
L'opération de *coupure* est l"inverse" de la *concaténation* : étants donnés un arbre 2-3-4 T et un élément x appartenant à T, la *concaténation* renvoie un couple (T_1, T_2) d'arbres 2-3-4 tels que T_1 contient tous les éléments de $T - \{x\}$ qui sont strictement inférieures à x et T_2 contient tous les éléments de $T - \{x\}$ qui sont strictement supérieures à x.
Le but de l'exercice est de définir les fonctions calculant la hauteur et le nombre de nœuds à profondeur k dans un arbre 2-3-4, puis de définir la concaténation de et la coupure de deux arbres 2-3-4.

Toutes ces fonctions seront implantées en utilisant les primitives sur les arbres 2-3-4 :

;EstVide : A2-3-4 → booleen
; EstVide(T) renvoie vrai si et seulement si T est l'arbre vide
;Degre :A2-3-4 → entier
; Degre(T) renvoie le nombre de sous-arbres à la racine de T
;Contenu :A2-3-4 → LISTE/de longueur 1 à 3/[entier]
; Contenu (T) renvoie la listes des éléments à la racine de T
*;EstDans : entier*LISTE[entier] → booleen*
; EstDans(e,L) renvoie vrai si et seulement si l'entier e appartient à la liste L
*;Elem : entier/de 1 à 3/ * A2-3-4 → entier*
; Elem(i,T) renvoie le ième élément à la racine de T et renvoie l'infini si le contenu de T a moins de 3 éléments
*;Ssab : entier/de 1 à 4/ * A2-3-4 → A2-3-4*
; Ssab(i,T) renvoie le ième sous-arbre à la racine de T, et renvoie l'arbre vide si le nombre de sous-arbres à la racine de T est inférieur à i

La figure suivante montre un arbre 2-3-4, T_0, de hauteur 2, ayant un nœud à profondeur 0 (la racine), quatre nœuds à profondeur 1 et dix nœuds à profondeur 2 (ces nœuds sont les *feuilles* de T_0).



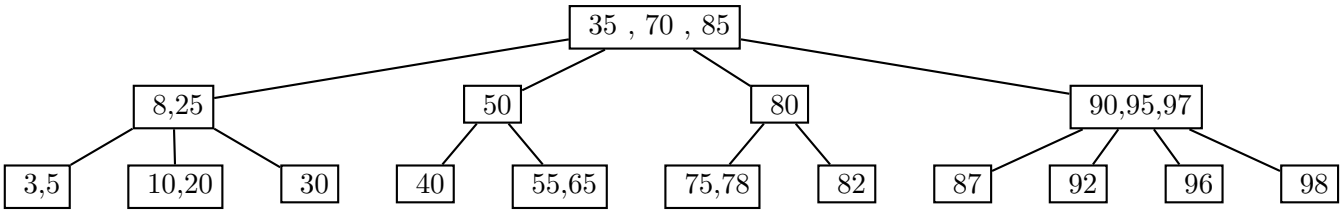
1- Écrire une définition de la fonction `hauteur`, qui renvoie la hauteur d'un arbre 2-3-4

2- Écrire une définition de la fonction `nb-noeuds-prof`, qui étant donnés un arbre 2-3-4 T et un entier k renvoie le nombre de nœuds à profondeur k dans T.

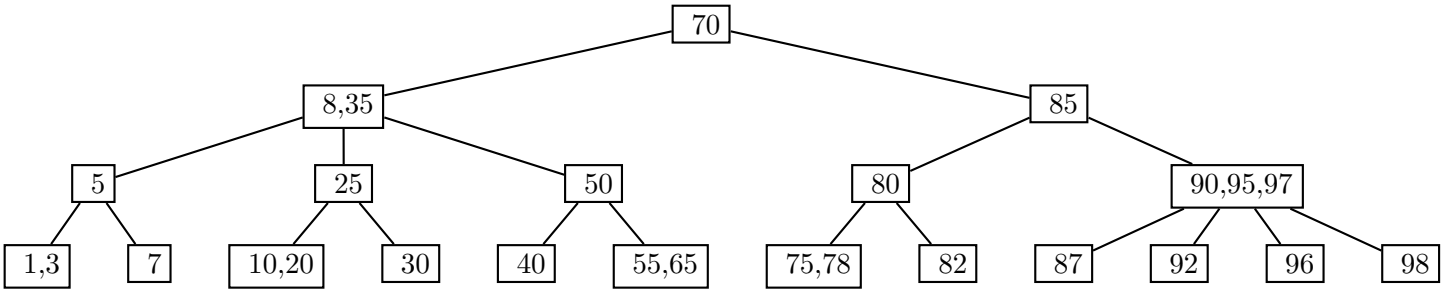
3- Calculer l'expression du nombre maximum et du nombre minimum de nœuds à profondeur k dans un arbre 2-3-4. En déduire un majorant de la hauteur d'un arbre 2-3-4 ayant n feuilles.

4- On considère deux arbres 2-3-4, T_1 et T_2, tels que toutes les étiquettes de T_1 sont strictement inférieures à toutes les étiquettes de T_2, et x une étiquette strictement comprise entre les étiquettes de T_1 et celles de T_2, (on pourra supposer que toutes les étiquettes sont différentes). L'algorithme de concaténation est le suivant :

- Si les hauteurs de T_1 et T_2 sont égales, il suffit de construire l'arbre ayant pour racine le nœud d'étiquette x, pour sous-arbre gauche T_1 et pour sous-arbre-droit T_2.
- Si h_1, la hauteur de T_1, est strictement inférieure à h_2, hauteur de T_2, soit U le sous-arbre dont la racine est à profondeur $h_2 - h_1$ sur la branche gauche de T_2. On concatène T_1, x et U en incluant x dans le nœud père de U, de telle sorte que toutes les feuilles de l'arbre résultant sont au même niveau.
  - Si le nœud père de U n'était pas un 4-nœud, on a terminé, comme le montre l'arbre ci-dessous, qui est la concaténation de l'arbre T_1 réduit à une feuille contenant les valeurs $(3, 5)$, de l'étiquette $x = 8$ et de l'arbre $T_2 = T_0$.



– Si le nœud père de U était un 4-nœud il faut rétablir la situation par des éclatements. Par exemple la concaténation de l'arbre $T_1 =< 5, (1,3), 7 >$, de l'étiquette $x = 8$ et de l'arbre $T_2 = T_0$ donne l'arbre ci-après.



– Si $h_1 > h_2$ on agit de manière analogue sur la branche droite de T_1.

Quel est le résultat de la concaténation de l'arbre $T_1 = T_0$, de l'étiquette $x = 100$ et de l'arbre T_2 réduit à une feuille contenant les valeurs $(105, 110, 120, 125)$.
Décrire l'algorithme de concaténation.
Déterminer le nombre maximum d'éclatements.

5- Il s'agit à présent de concaténer deux arbres 2-3-4, T_1 et T_2, tels que toutes les étiquettes de T_1 sont strictement inférieures à toutes les étiquettes de T_2, *sans l'intermédiaire* d'une étiquette x.
Expliquer quels sont les différents cas de figure possibles et comment étendre l'algorithme précédent.

6- Décrire l'algorithme de *coupure* et analyser sa complexité.

3 Coût amorti [5 points]

Cet exercice présente une implémentation de pile qui utilise un peu de mémoire centrale et beaucoup d'espace disque. L'espace disque est partitionné en *pages*, pouvant chacune contenir p éléments. On peut lire et écrire les éléments *individuellement* lorsqu'ils sont en mémoire centrale, avec un coût $O(1)$; mais lorsque les éléments sont sur disque, on y accède *par page*, avec un coût $O(p)$.
On considère une pile qui stocke la première page de ses éléments (haut de pile) en mémoire centrale, et la suite sur disque ; ll y a donc un coût $O(p)$ lorsque le sommet de pile change de page.

1- Montrer que pour une suite de n opérations *empiler* ou *dépiler*, le nombre maximum d'accès disque est $O(n)$ et le coût maximum $O(np)$. Décrire une suite de n opérations qui atteint ce coût maximum.

2- Montrer que si on peut stocker 2 pages en mémoire centrale, alors il est possible d'implanter les opérations *empiler* et *dépiler* avec un coût amorti $O(1)$.

4 NP-Complétude [5 points]

1- Supposons que l'on dispose d'un algorithme pour résoudre le problème de la clique : étant donné un graphe G et un entier k, l'algorithme $A(G, k)$ décide si G a une clique de taille k. Donner alors un algorithme qui calcule, en utilisant seulement des appels à l'algorithme A, les sommets d'une k-clique dans un graphe (si elle existe).

2- Montrer que le problème de décision suivant est NP-Complet. Étant donnés deux graphes G_1, G_2 et un entier k, déterminer s'il existe un graphe avec au moins k arêtes qui est à la fois un sous-graphe de G_1 et un sous-graphe de G_2.

3- Supposons que l'on dispose d'un algorithme B pour résoudre le problème du plus grand sous-graphe commun de la question précédente. Donner alors un algorithme qui calcule, en utilisant seulement des appels à l'algorithme B, un sous-graphe de k arêtes (s'il existe) apparaissant dans G_1, G_2.