Algorithmique et structures de données : examen de première session

Ensimag 1A

Année scolaire 2012–2013

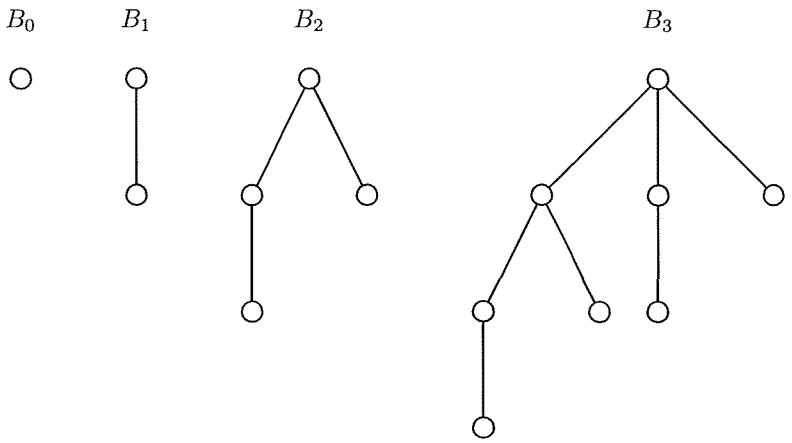
Consignes générales :

- Durée : 3 heures.
- Calculatrices, portables et tous instruments électroniques interdits. Livres interdits. Autres documents autorisés.
- Le barème est donné à titre indicatif.
- Les exercices sont indépendants et peuvent être traités dans le désordre.
- La syntaxe Ada ne sera pas un critère déterminant de l'évaluation des copies. En d'autres termes, les correcteurs n'enlèveront pas de point pour une syntaxe Ada inexacte mais compréhensible (pensez à bien commenter votre code !).
- **Merci d'indiquer votre numéro de groupe de TD et de rendre votre copie dans le tas correspondant à votre groupe.**

1 Tas binomiaux (12 points)

Rappelons qu'un arbre au sens informatique du terme possède une *racine* r, pour un sommet u de l'arbre, tous les sommets w tels que u est le dernier sommet avant w sur l'unique chemin de r à w s'appellent les *fils* de u. L'arbre est dessiné par niveaux, tous les sommets à la même distance de la racine sont sur le même niveau et les fils de chaque sommet sont ordonnés de gauche à droite.
Un arbre binômial B_k est un arbre défini récursivement comme suit :
- B_0 est l'arbre formé d'une racine et pas d'autres sommets.
- B_k est composé de deux arbres B_{k-1} qui sont liés de la façon suivante : la racine de l'un est le fils le plus à gauche de la racine de l'autre.

La figure suivante donne B_0, B_1, B_2 et B_3.



Question 1 : Dessiner B_4. **(0.5 point)**
Question 2 : (On pourra considérer comme acquises les propriétés suivantes pour la suite du problème si on n'a pas réussi à faire les démonstrations qui sont toutes très simples) Montrer que pour B_k :

1. Il y a 2^k sommets. **(0.5 point)**
2. La hauteur (ou profondeur) est de k. **(0.5 point)**
3. Il y a exactement C_k^i sommets au niveau i. (Les niveaux sont comptés de haut en bas, le niveau 0 contient la racine et le niveau k est celui le plus bas.) **(0.5 point)**
4. La racine possède k fils. En numérotant les fils de la droite vers la gauche en commençant par 0 (donc le fils le plus à droite est numéroté 0 et le fils le plus à gauche est numéroté $k-1$), le fils numéro i est la racine d'un B_i. **(0.5 point)**

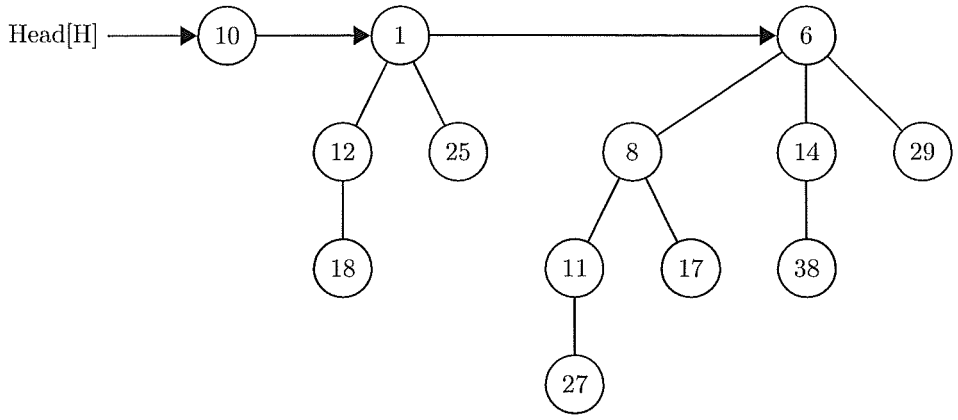
Question 3 : Déduire des propriétés précédentes que si n est le nombre de sommets d'un arbre binomial alors aucun sommet n'a plus de $\log n$ fils. **(1 point)**

Un *tas binômial* H (*binomial heap* en anglais) est un ensemble d'arbres binômiaux tels que les propriétés suivantes sont satisfaites :

1. *Min-ordonné* : Dans chaque arbre binômial de H, la valeur attachée à chaque sommet (appelée dans la suite clé) est inférieure ou égale à celle attachée à ses fils (on aurait pu définir max-ordonné)

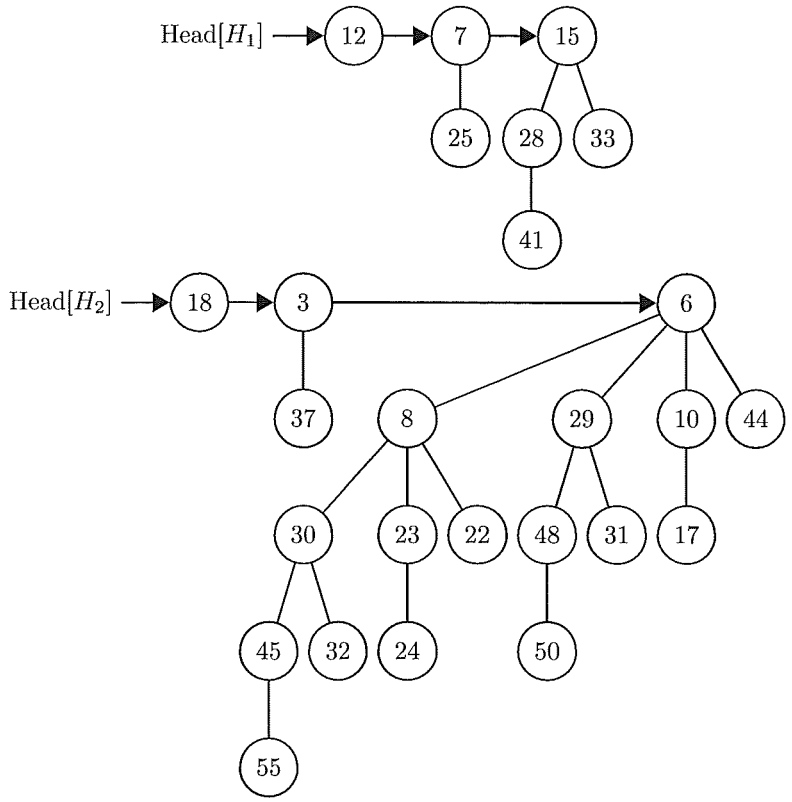
2. Pour tout entier k, il y a **au plus un** arbre binômial dans H de racine de degré k. (Au plus un veut dire au maximum un).

On utilisera la propriété suivante, qui est une conséquence de la deuxième condition ci-dessus, sans la démontrer : Dans un tas binômial H contenant n sommets en tout, il y a au plus $\lfloor \log n \rfloor + 1$ arbres. (Pour voir pourquoi, écrivons n sous forme binaire $n = \sum_{i=0}^{\lfloor \log n \rfloor} b_i 2^i$, on sait qu'un arbre binomial B_i possède 2^i sommets, donc l'arbre binomial B_i apparaîtra dans le tas H si et seulement si $b_i = 1$)
Dans un tas binômial H chaque nœud est une structure avec un lien vers son fils aîné, un lien vers son frère cadet et un autre vers son père. Dans le cas d'absence de tels parents le lien est $NULL$. De plus les racines des arbres sont dans une liste chaînée de gauche à droite (on peut utiliser pour cela le lien frère cadet des racines). La première racine est donnée par $head[H]$. Dans ce qui suit on supposera que les racines sont ordonnées de gauche à droite selon leur degré (nombre de fils), c'est ce qui est fait dans la figure suivante.



Question 4 : Ecrire l'algorithme de recherche de la clé minimum et déduire de ce qui précède que trouver la clé minimum se fait en $O(\log n)$. **(1 point)**
Question 5 : La base de toutes les opérations sur les tas binomiaux est la fusion de deux tas binomiaux. La figure ci-dessous donne deux tas binomiaux H_1 et H_2
Pour fusionner deux tas binômiaux on met les arbres des deux tas dans une même liste ordonnée par degré croissant des racines.

1. Dire pourquoi en général cela ne donne pas un tas binômial. **(0.5 point)**
2. Dans l'exemple qui vous est proposé, par des opérations que vous décrirez soigneusement vous ramener à un tas binômial. Vous pouvez simplifier les figures dans la mesure où on comprend bien ce que vous faites. Les figures ne sont qu'un support des explications mais ne les remplacent pas. **(1 point)** (Aide : essayez de maintenir la propriété que les arbres de la liste sont en ordre croissant des degrés des racines)
3. Donner les grandes lignes d'un algorithme de fusion de deux tas binômiaux. **(2 points)**
4. Quelle est la complexité de l'algorithme précédent en fonction du nombre de sommets final $n = n_1 + n_2$. **(1 point)**



Question 6 : En utilisant l'opération de fusion de deux tas comme une fonction, écrire l'algorithme d'insertion d'un nouvel élément dans un tas binômial. Complexité ? **(1.5 point)**
Question 7 : Ecrire l'algorithme d'extraction (suppression) de l'élément de clé minimum dans un tas binômial. (Cela utilise à nouveau la fusion de deux tas, mais aussi la propriété 4 de la question 2). Complexité ? **(1.5 point)**

2 Algorithme des *4 russes* (8 points)

L'idée de cet exercice est d'analyser l'algorithme dit des *4 russians* qui utilise une technique de décomposition pour multiplier deux matrices carrées booléennes. Cette opération est en particulier utile pour le calcul de la fermeture transitive d'un graphe représenté par sa matrice d'adjacence.
On cherche à concevoir un algorithme efficace pour multiplier deux matrices booléennes carrées A et B de dimension n ($C = A \times B$). Les opérations de base s'entendent ici au sens booléen, c'est-à-dire, la multiplication élémentaire comme un $\wedge$ logique et l'addition comme un $\vee$: $C_{i,j} = \vee_{k=1,\ldots,n} A_{i,k} \wedge B_{k,j}$
Question 8 : On considère le graphe à 4 sommets de la figure 1. Construisez sa matrice d'adjacence A et calculez A^2 ($A \times A$). **(0.5 point)**
Question 9 : L'algorithme usuel de produit de matrices correspond à écrire les n^2 opérations de produits des lignes de A par les colonnes de B.
Écrivez (avec un pseudo-code de bas niveau) la version de l'algorithme qui parcourt la

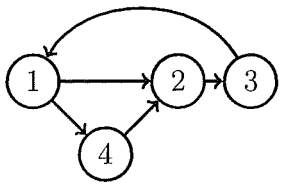


FIGURE 1 – Graphe exemple

matrice A par colonnes et B par lignes. Plus précisemment, ceci signifie que dès que l'on accède pour la première fois à une cellule de la i$^{\text{ème}}$ colonne de A, on s'interdit les accès aux colonnes d'indices inférieurs à i. Ce principe s'applique également aux lignes de B. **(2 points)**

Question 10 : L'idée de base de l'algorithme des 4 russes est de partitionner la matrice A en groupes de tailles égales de λ colonnes consécutives et B en groupes de λ lignes consécutives. Pour simplifier, on supposera ici que λ divise n. On notera A_i la matrice n par λ correspondante et B_i la matrice λ par n. La figure 2 illustre la décomposition de A et B.

Donner l'expression du produit $A \times B$ en utilisant A_i et B_i. **(0.5 point)**

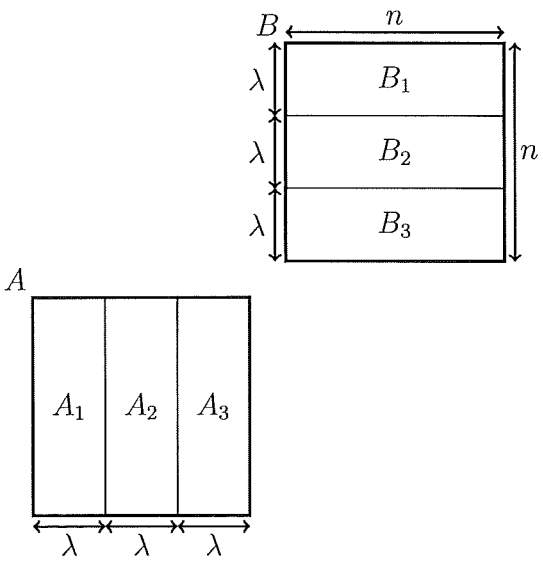


FIGURE 2 – Décomposition des matrices A et B

Question 11 : On se propose d'utiliser une fonction `Produit_Ligne_Matrice` prenant en entrée une ligne d'une matrice A_i ainsi qu'une matrice B_i (plutôt que de donner toute la matrice B_i en paramètre, il est possible de donner uniquement l'indice i et de lire ensuite les données dans un tableau global à 3 dimensions par exemple). Dans cette question on cherche à calculer $C_i = A_i \times B_i$ **pour un i fixé**. La fonction `Produit_Ligne_Matrice` est appellée n fois pour construire chacune des n lignes de C_i. En réalité de nombreux appels sont redondants, c'est à dire que la fonction est appellée plusieurs fois avec des paramètres

exactement identiques. Proposez une borne supérieure sur le nombre maximal d'appels différents possibles pour réaliser le calcul de C_i. **(1 point)**

Question 12 : On se propose de réduire le coût de l'algorithme en évitant de réaliser deux fois le même calcul à l'aide d'une technique appellée mémoïsation. Pour ce faire, on se propose de stocker tous les résultats des appels à la fonction `Produit_Ligne_Matrice` dans un dictionnaire D. Chaque entrée possible (une ligne + une matrice) correspond à une clef à laquelle se trouve associée le résultat de la fonction. On ajoute en début de la fonction `Produit_Ligne_Matrice` une conditionelle vérifiant si une valeur apparait dans D pour la clef correspondant aux arguments et retournant immédiatement le résultat s'il est déjà calculé. Dans le cas contraire, la fonction s'exécute normalement puis sauve le nouveau résultat dans D avant de le renvoyer.
Expliquez le principe d'un algorithme efficace (grâce à la mémoïsation) calculant $A \times B$. Précisez le type de dictionnaire que vous utilisez. **(2 points)**

Question 13 : Evaluez son coût au pire cas en fonction de n et λ. (détaillez le calcul) **(1 point)**

Question 14 : Proposez une valeur de λ permettant d'obtenir un temps de calcul bien meilleur que le produit classique (justifiez). **(0.5 point)**

Question 15 : La fermeture transitive d'un graphe G peut être stockée par une matrice booléenne F contenant sur la case $F(i,j)$ un booléen vrai s'il est possible d'aller de i à j et faux sinon. Si A est la matrice d'adjacence du graphe G, alors $F = (A + I)^n$ où I désigne la matrice identité et n le nombre de sommets de G. Proposez un algorithme rapide (donnez l'idée) pour calculer la fermeture transitive d'un graphe. Quel est son coût au pire cas ? **(0.5 point)**