Examen – Session 1

Analyse d'Algorithmes et Programmation

30 avril 2018

Durée : 2 heures

Les seuls documents autorisés sont les notes de cours.
L'utilisation d'un appareil électronique est proscrit pendant toute la durée de l'épreuve.
Le barème est indicatif.

Exercice 1 — Tri et arbre [2 points]

Question 1.1 [1 point]
Donner un algorithme pour trier une liste de nombres qui utilise un arbre binaire de recherche.

> **Solution:** Il suffit d'insérer un à un les éléments de la liste dans un arbre binaire de recherche, puis de retourner le parcours infixe de cet arbre.

Question 1.2 [1 point]
Quel est le temps d'exécution de votre algorithme ? Comparer avec les autres algorithmes de tri connus.

> **Solution:** Si h est la hauteur de l'arbre, alors la complexité est en $O(nh)$. En moyenne, on obtient du $O(n\log(n))$ et du $O(n^2)$ dans le pire cas, celui d'un tableau déjà trié.

Exercice 2 — Arbres binaires de recherche [4.5 points]

Question 2.1 [1 point]
Soit T un arbre binaire de recherche contenant $n \geq 1$ nœuds. Donner des bornes sur la hauteur de l'arbre T.

> **Solution:**
> $$\lfloor \log_2(n) \rfloor \quad \leq \quad h \quad \leq \quad n-1.$$

Question 2.2 [1 point]

Réciproquement, soit T un arbre binaire de recherche de hauteur $h(T) \geq 0$. Quels sont les nombres minimaux et maximaux de nœuds contenus dans l'arbre T ?

Solution:

$$h+1 \quad \leq \quad n \quad \leq \quad 2^{h+1}-1.$$

Question 2.3 [1 point]

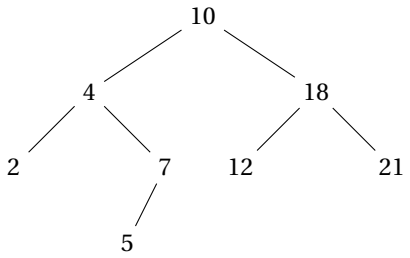
Soit la liste de nombres suivante : $\{1,2,3,4,5,6,7\}$. La hauteur de l'arbre binaire de recherche construit d'après cette liste dépend de l'ordre d'insertion dans l'arbre. Donner un ordre d'insertion pour chacune des hauteurs possibles.

Solution:

- $h = 2$: 4, 2, 1, 3, 6, 5, 7
- $h = 3$: 4, 3, 2, 1, 6, 5, 7
- $h = 4$: 5, 4, 3, 2, 1, 6, 7
- $h = 5$: 6, 5, 4, 3, 2, 1, 7
- $h = 6$: 7, 6, 5, 4, 3, 2, 1

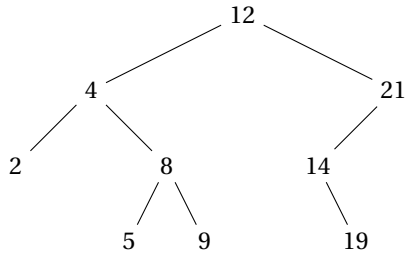
Question 2.4 [1.5 points]

Soit l'arbre binaire de recherche T ci-contre. Appliquez-lui **dans l'ordre** les instructions suivantes et retournez le résultat.



- INSERER(T, 14)
- SUPPRIMER(T, 18)
- INSERER(T, 19)
- INSERER(T, 9)
- INSERER(T, 8)
- SUPPRIMER(T, 7)
- SUPPRIMER(T, 10)

Solution:



Exercice 3 — Table de hachage [3.5 points]

J'ai à représenter un sous-ensemble de 17 éléments de $\{0,\ldots,50\}$. J'aimerais utiliser une table à adressage direct mais je n'ai à disposition qu'une table à 11 entrées $T[0,\ldots,10]$.

Question 3.1 [1 point]
Expliquer en quoi l'adressage direct est impossible ici.

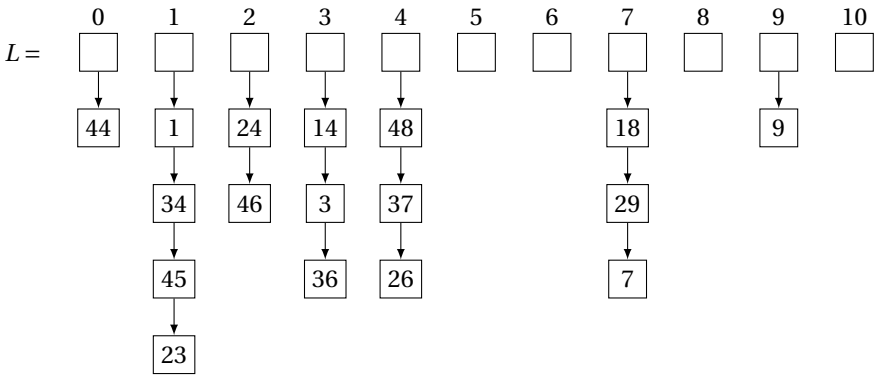
Solution: Pour utiliser l'adressage direct, il faudrait une table de 51 entrées.

Question 3.2 [1 point]
Je vais donc utiliser une fonction de hachage de la forme $h(k) = k \bmod m$. Expliquer en quoi prendre $m = 11$ est un bon choix ?

Solution: Tout d'abord, ce choix est en adéquation avec la taille de la table. Ensuite, 11 est premier et est éloigné d'une puissance de 2.

Question 3.3 [1.5 points]
Donner l'état de la table T correspondant à l'ensemble $\{46,26,23,37,48,45,44,34,1,36,9,7,3,14,29,18,24\}$, pour la fonction définie à la question précédente. Les collisions seront résolues par chaînage.

Solution:



Question 3.4 [1 point]
Quel est l'élément dont la recherche a un coût maximal ? En notant c le coût d'un accès à un pointeur quel est le temps exact de cette recherche ?

Solution: L'élément dont la recherche possède un coût maximal est 23. Ce coût est de $4c$.

Exercice 4 — Programmation dynamique [3 points]

Question 4.1 [1.5 points]
Modifier l'algorithme COUPER-BARRE-ASC pour y ajouter le prix de coupe, fixé à 1 (c'est-à-dire que chaque coupe effectuée coûte 1, à retirer du gain total). La sortie de votre algorithme doit retourner à la fois les tableaux des revenus r et des positions de coupe s.

```
COUPER-BARRE-ASC(p, n)
 1: r = tab[0, ..., n]
 2: r[0] = 0
 3: for j = 1 to n do
 4:     q = −∞
 5:     for i = 1 to j do
 6:         q = max(q, p[i] + r[j − i])
 7:     end for
 8:     r[j] = q
 9: end for
10: return r[n]
```

Solution:

```
COUPER-BARRE-AVEC-COUT(p, n)
 1: r = tab[0, ..., n]
 2: s = tab[0, ..., n]
 3: r[0] = 0
 4: s[0] = 0
 5: for j = 1 to n do
 6:     q = p[j]
 7:     s[j] = j
 8:     for i = 1 to j − 1 do
 9:         if q < p[i] + r[j − i] − 1 then
10:             q = p[i] + r[j − i] − 1
11:             s[j] = i
12:         end if
13:     end for
14:     r[j] = q
15: end for
16: return r, s
```

Question 4.2 [1.5 points]
Donner la sortie de ce nouvel algorithme pour $n = 10$ et le tableau de tarifs suivant :

longueur i	1	2	3	4	5	6	7	8	9	10
prix p_i	1	5	8	9	10	17	17	20	24	30

Solution:

$$r = [0,1,5,8,9,12,17,17,21,24,30]$$
$$s = [0,1,2,3,4,2,6,7,2,9,10]$$

Une autre possibilité pour le s est :

$$r = [0,1,5,8,9,12,17,17,21,24,30]$$
$$s = [0,1,2,3,2,3,6,6,6,6,10]$$

Exercice 5 — Algèbre linéaire [3 points]

Question 5.1 [1 point]
Qu'est-ce que ω, l'exposant de l'algèbre linéaire ? Donnez-en un encadrement entre 2 entiers en expliquant votre choix.

Solution: ω est la plus petite constante c telle qu'on puisse multiplier deux matrices $n \times n$ en $O(n^c)$. Trivialement, on a $2 \leq \omega \leq 3$, puisque il y a au moins n^2 coefficients à écrire et que l'algorithme naïf est en $O(n^3)$.

Question 5.2 [1 point]
Donner deux valeurs précises pour ω, accompagnées du nom de l'algorithme associé.

Solution: Algo naïf : $\omega = 3$.
Algo de Strassen : $\omega = \log_2(7) \approx 2.807$.

Question 5.3 [1 point]
Montrer, en raisonnant dans les deux sens, que calculer le carré d'une matrice de taille $n \times n$ a aussi une complexité en $O(n^\omega)$.

Solution:

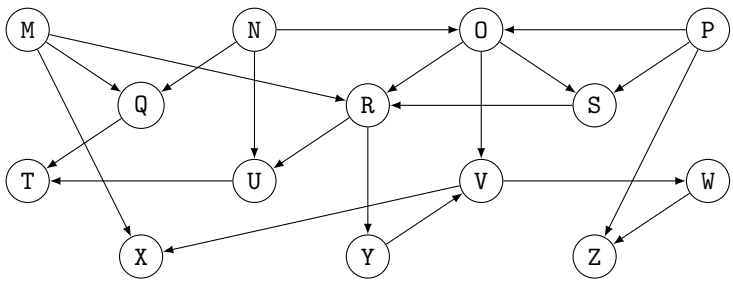
- Si on sait multiplier, on peut utiliser cet algorithme pour calculer le carré.
- Si on sait calculer le carré, on peut obtenir

$$\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}^2 = \begin{pmatrix} AB & 0 \\ 0 & AB \end{pmatrix}$$

  et on retrouve facilement le produit AB.

Exercice 6 — Graphes [6.5 points]

Question 6.1 [1.5 points]
Voici un graphe orienté non-pondéré. Appliquez l'algorithme de tri-topologique et retournez la liste des sommets triés. Vous ferez apparaître sous forme d'un tableau les données intermédiaires. On suppose que lorsque l'on a le choix, l'ordre lexicographique est respecté.

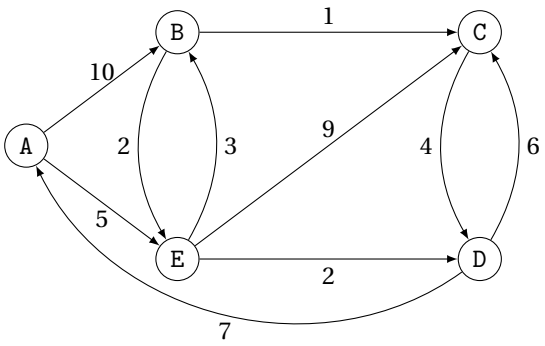


Solution:

nœud u	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$u.d$	1	21	22	27	2	6	23	3	7	10	11	15	9	12
$u.f$	20	26	25	28	5	19	24	4	8	17	14	16	18	13

Nœuds triés : P N O S M R Y V X W Z U Q T.

Question 6.2 [1 point]
Construire la matrice d'adjacence du graphe suivant :



Solution: En numérotant les sommets selon l'ordre lexicographique,

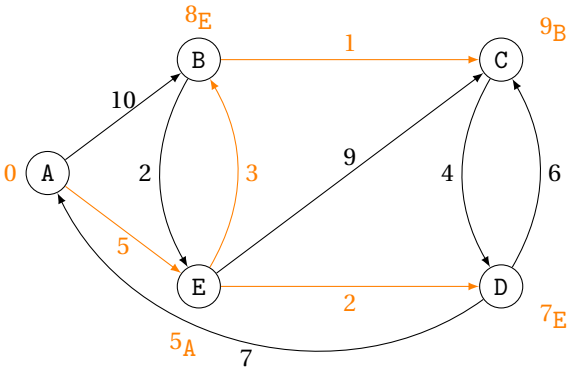
$$\begin{pmatrix} 0 & 10 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 4 & 0 \\ 7 & 0 & 6 & 0 & 0 \\ 0 & 3 & 9 & 2 & 0 \end{pmatrix}$$

Question 6.3 [1 point]
Quelles sont les différences entre matrices et listes d'adjacence ? Citer un cas pratique où l'un des deux choix est meilleur que l'autre.

Solution: Les listes demandent moins de mémoire alors que les matrices permettent une recherche d'arête plus rapide. Dans le cas d'un graphe avec peu d'arêtes, il faut privilégier les listes.

Question 6.4 [1.5 points]
Appliquer l'algorithme de DIJKSTRA au graphe précédent et expliquer le résultat.

Solution:



Question 6.5 [1.5 points]

Écrire un algorithme PLUS-COURT-CHEMIN(u, v) qui affiche les étapes d'un plus court chemin de u à v de manière lisible. On suppose que DIJKSTRA(G, u) a déjà été appliqué au graphe.

Solution:

```
PLUS-COURT-CHEMIN(u, v)
 1: if v == u then
 2:     print u
 3: else
 4:     if v.π == NIL then
 5:         print "no path"
 6:     else
 7:         PLUS-COURT-CHEMIN(u, v.π)
 8:         print v
 9:     end if
10: end if
```