

NOM :

Prénom :

- tous les documents (poly, slides, TDs, livres...) sont autorisés.
- les réponses doivent toutes tenir dans les cases prévues dans l'énoncé (pas de feuilles supplémentaires).
- le code des algorithmes/fonctions pourra être donné soit en C soit en pseudo-code. En C la syntaxe n'a pas besoin d'être exacte, mais le code doit être présenté clairement (accolades, indentation, écriture lisible...). En pseudo-code, aucune syntaxe n'est imposée, mais tout devra être suffisamment détaillé pour ne pas laisser de place aux ambiguïtés (écrire "parcourir les sommets du graphe G" n'est pas assez précis, mais "parcourir les voisins du sommet S" est correct).
- le barème actuel de ce contrôle aboutit à une note sur 105 points. La note finale sur 20 sera dérivée de la note sur 105 grâce à une fonction croissante qui n'est pas encore définie (cela ne sert à rien de la demander).
- n'oubliez pas de remplir votre nom et votre prénom juste au dessus de ce cadre.

**Exercice 1 : QCM**

(10 points)

Chaque bonne réponse rapporte 1 point. Chaque mauvaise réponse enlève 1 point. Ne répondez pas au hasard, la note totale peut être négative !

**1.a]** Laquelle de ces structures de données ne peut pas s'implémenter efficacement avec un tableau :

- une pile,             une file,             un arbre général,     un tas.

**1.b]** Pour stocker quelques centaines d'éléments dans une table de hachage, laquelle de ces valeurs de  $k$  (le nombre de hachés différents) sera le meilleur choix pour avoir la recherche la plus rapide possible sans pour autant gaspiller de mémoire ?

- 10                     100                     1000                     10000

**1.c]** Quelle sera la hauteur totale d'un arbre binaire de recherche dans lequel on insère successivement les éléments 7, 3, 9, 5, 8, 12, 4 ?

- 1                     2                     3                     4

**1.d]** Quelle sera la hauteur totale d'un arbre AVL dans lequel on insère successivement les éléments 7, 3, 9, 5, 8, 12, 4 ?

- 1                     2                     3                     4

**1.e]** On part d'un graphe orienté contenant 8 sommets et 7 arcs. Au maximum, combien d'arcs pourra contenir sa *fermeture transitive réflexive* ?

- 8                     36                     50                     64

**1.f]** Des éléments sont placés dans un arbre binaire de recherche. On affiche le contenu de cet arbre par un parcours *infixe*. Lequel de ces affichages est possible ?

- 4,1,3,6,5,2       1,2,3,4,5,6       3,1,4,2,5,6       2,1,5,3,4,6

**1.g]** Des éléments sont placés dans un arbre binaire de recherche. On affiche le contenu de cet arbre par un parcours *préfixe*. Lequel de ces affichages est possible ?

- 4,1,3,6,5,2       1,2,3,5,6,4       3,1,4,2,5,6       2,1,5,3,4,6

---

```
1 int func(int n) {
2   int i = 0;
3   int j = 0;
4   while (i <= n) {
5     i = 2*j + i + 1;
6     j++;
7   }
8   return j-1;
9 }
```

---

**1.h]** Quelle est la complexité de la fonction `func` ?

- $O(\log n)$ ,        $O(\sqrt{n})$ ,        $O(n)$ ,        $O(\frac{n}{\log n})$ .

**1.i]** Combien vaut `func(9999)` ?

- 9,       99,       999,       4999.

**1.j]** Parmi les valeurs suivantes, laquelle vaut-il mieux utiliser pour le modulo de l'algorithme de Rabin-Karp avec modulo si on utilise un processeur 64 bits ?

- 64,       65521,        $2^{32} - 1$ ,        $2^{63}$ .

**Exercice 2 :** Plus courts chemins dans un graphe pondéré (30 points)

On cherche ici à calculer des plus courts chemins dans des graphes pondérés non-orientés. Chaque arête possède un poids et, partant d'un sommet donné, on veut calculer les plus courts chemins (les chemins dont la somme du poids des arêtes est minimal) vers tous les autres sommets du graphe. Pour simplifier, on considère que le graphe est connexe (il existe un chemin qui relie n'importe quelle paire de sommets).

(2 pts) **2.a]** Dans cet exercice, tous les algorithmes doivent calculer des plus courts chemins en partant d'un sommet donné, pas tous les plus courts chemins du graphe. Pour quelle représentation de graphe opteriez-vous donc *a priori* ? Pourquoi ?

- (3 pts) **2.b]** Considérons tout d'abord le cas simple où toutes les arêtes du graphe ont un poids égal à 1. Comment calculer les plus courts chemins partant d'un sommet  $u$  donné vers tous les autres sommets du graphe? Attention, on ne veut pas simplement la longueur du chemin, on veut aussi pouvoir facilement retrouver le chemin à suivre pour aller de  $u$  à un sommet  $v$ . Quelle est la complexité de cet algorithme (en fonction de  $|A|$  et  $|S|$ , le nombre d'arêtes et de sommets du graphe)?

On considère à partir de maintenant que les arêtes ont un poids quelconque, strictement positif. On part d'un sommet  $u$  qui est donc à distance 0. Le principe de l'algorithme est de séparer les sommets du graphe en deux catégories : les sommets optimisés, pour lesquels on connaît déjà le plus court chemin et sa longueur, et les sommets non-optimisés. Au départ, seul le sommet  $u$  est optimisés et tous les autres sont non-optimisés.

- (2 pts) **2.c]** Parmi les voisins du sommet  $u$ , l'un peut tout de suite être classé dans les sommets optimisés? Lequel, comment et pourquoi?

- (1 pts) **2.d]** De façon générale, l'un des voisins des sommets optimisés (parmi les sommets non-optimisés) peut toujours être optimisé. Lequel, comment?


- (7 pts) **2.e]** Écrivez un algorithme qui calcule les plus courts chemins partants d'un sommet  $u$  dans un graphe  $G$ . En particulier, détaillez bien la structure de données dans laquelle vous rangez les sommets non-optimisés, comment elle est initialisée et comment elle est mise à jour. Encore une fois on veut connaître la longueur du chemin de  $u$  à n'importe quel sommet  $v$ , mais on veut aussi pouvoir retrouver ce chemin. Quelle est la complexité de votre algorithme? Pensez-vous que l'on puisse l'améliorer?

On s'intéresse maintenant au cas où les arêtes du graphe peuvent aussi avoir un **poids négatif**.

- (2 pts) **2.f]** Dans l'algorithme précédent, on pouvait déjà connaître certains plus courts chemins avant d'avoir observé toutes les arêtes du graphe. Ce n'est plus le cas ici, pourquoi ?

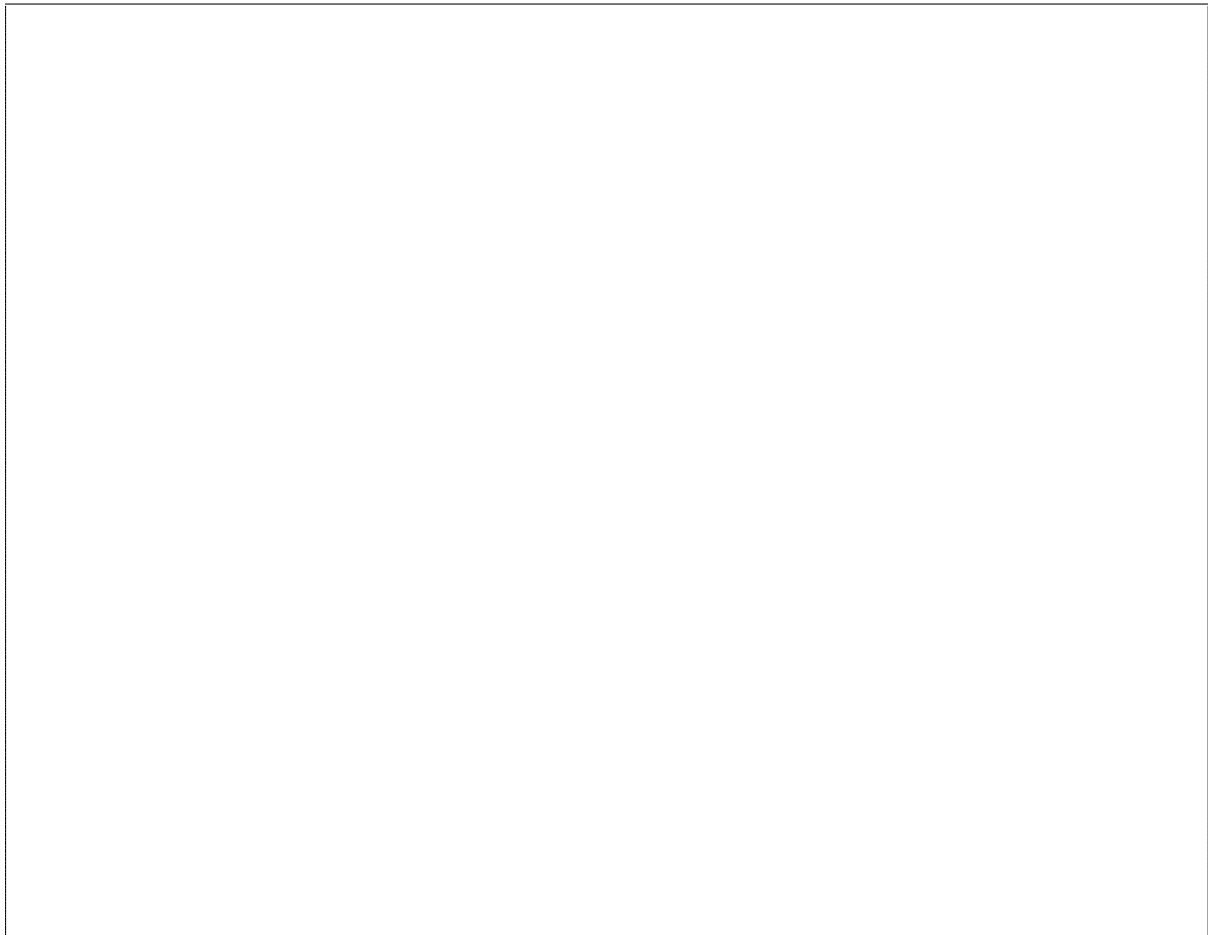
- (2 pts) **2.g]** La présence de cycles ne pose aucun problème si les poids de toutes les arêtes du graphe sont positifs. Si certaines arêtes ont un poids négatif, dans quel cas les cycles posent-ils un problème pour la recherche de plus courts chemins ?

- (7 pts) **2.h]** Pour chercher les plus courts chemins partant de  $u$  il est cette fois nécessaire de tester "tous les chemins possibles". En fait, on regarde uniquement les chemins de longueur  $|S| - 1$  ou moins. L'algorithme que vous allez écrire doit donc initialiser les distances de tous les sommets du graphe, puis  $|S| - 1$  fois à la suite, mettre à jours ces distances par un parcours des arêtes du graphe. Écrivez un tel algorithme, expliquez brièvement pourquoi il fonctionne et donnez sa complexité.





(4 pts) **2.i]** Une fois cet algorithme exécuté, comment peut-on vérifier l'absence de cycles "nuisibles" ?



### Exercice 3 : File d'attente avec gestion de priorité

(20 points)

Une entreprise décide d'informatiser la gestion de ses commandes/productions. Ils ont besoin de pouvoir stocker la référence des commandes en attente et de pouvoir traiter ces commandes en fonction de leur priorité. Chaque commande est référencée par un numéro de commande unique (un entier) et une priorité.

Dans cet exercice on considère que toutes les structures de données du cours (pile, file, liste chaînée, tas, table de hachage, arbre binaire de recherche...) sont déjà implémentées et accessibles par leur fonctions standards (`push`, `pop`, `insert`, `delete`, `is_empty`...).

- (3 pts) **3.a]** Le responsable de l'entreprise décide tout d'abord de pouvoir gérer des commandes ayant des priorités représentées par un entier quelconque (un `int` de 32 bits). Il veut pouvoir :
- ajouter de nouvelles commandes ayant une priorité quelconque à la file d'attente,
  - retirer la commande ayant la priorité la plus élevée de la file d'attente et récupérer son numéro de commande.

Quelle structure de données utiliseriez-vous pour gérer efficacement cette file d'attente ?

Quel serait alors la complexité (en fonction du nombre de commandes  $n$  en attente) des opérations d'insertion et d'extraction d'un élément ?

- (3 pts) **3.b]** Après quelques mois d'utilisation de leur nouveau système, le gestionnaire de la file d'attente se rend compte qu'il n'utilise en fait que 3 priorités différentes : 2 pour les priorités hautes, 1 pour les priorités moyennes et 0 pour les priorités basses. Cependant il aimerait aussi que des commandes ayant la même priorité soient toujours traitées dans l'ordre de leur arrivée (en FIFO).

La structure précédente convient-elle toujours ? Pourquoi ?

- (4 pts) **3.c]** Quelle structure de donnée utiliseriez-vous afin de pouvoir ajouter/extraire une commande *en temps constant* (indépendant du nombre de commandes en attente) ?

- (6 pts) **3.d]** Écrivez les fonctions `void ajouter(int prio, int num)` et `int extraire()` qui vont respectivement ajouter la commande numéro `num` avec la priorité `prio` dans la file d'attente et extraire le numéro de commande de la prochaine commande à traiter (tout en supprimant cette commande de la file d'attente).



(4 pts) **3.e]** Une entreprise voisine ayant entendu parlé du succès de la gestion informatique des commandes se décide à installer un système similaire dans ses locaux. En revanche ses besoins sont un peu différents : elle ne veut pas avoir à attribuer des priorités à chaque commande mais préfère définir des priorités par client du type “les commandes du client A doivent toujours être traitées avant les commandes du client B”. Cette entreprise possède un ensemble de clients donné et définit les règles une fois pour toute. Quelle solution proposeriez-vous pour gérer une file d’attente de commandes tout en respectant ces règles ? Quand une nouvelle commande arrive, que faut-il faire ?

**Exercice 4 : Implémentation d'un tas avec un arbre binaire**

(20 points)

Nous avons vu en cours comment implémenter un tas à l'aide d'un tableau. Cependant, il est aussi possible d'implémenter un tas à l'aide d'un arbre binaire, tout en conservant les complexités d'insertion/suppressions obtenues avec un tableau.

- (1 pts) **4.a]** Pour quelle raison pourrait-on avoir envie d'implémenter un tas à l'aide d'un arbre binaire si cela n'améliore pas la complexité des opérations par rapport à une implémentation avec un tableau ?

Pour des questions d'efficacité, l'implémentation d'un tas avec un arbre binaire nécessite de modifier légèrement la structure classique d'arbre binaire : il faut ajouter un compteur du nombre de nœuds dans l'arbre et chaque nœud doit aussi contenir un pointeur vers son père. On utilise donc une variable globale `int num` égale au nombre d'éléments dans le tas et la structure de nœud est la suivante :

---

```
1 int num;
2
3 struct node {
4     int val;
5     struct node* left;
6     struct node* right;
7     struct node* father;
8 };
```

---

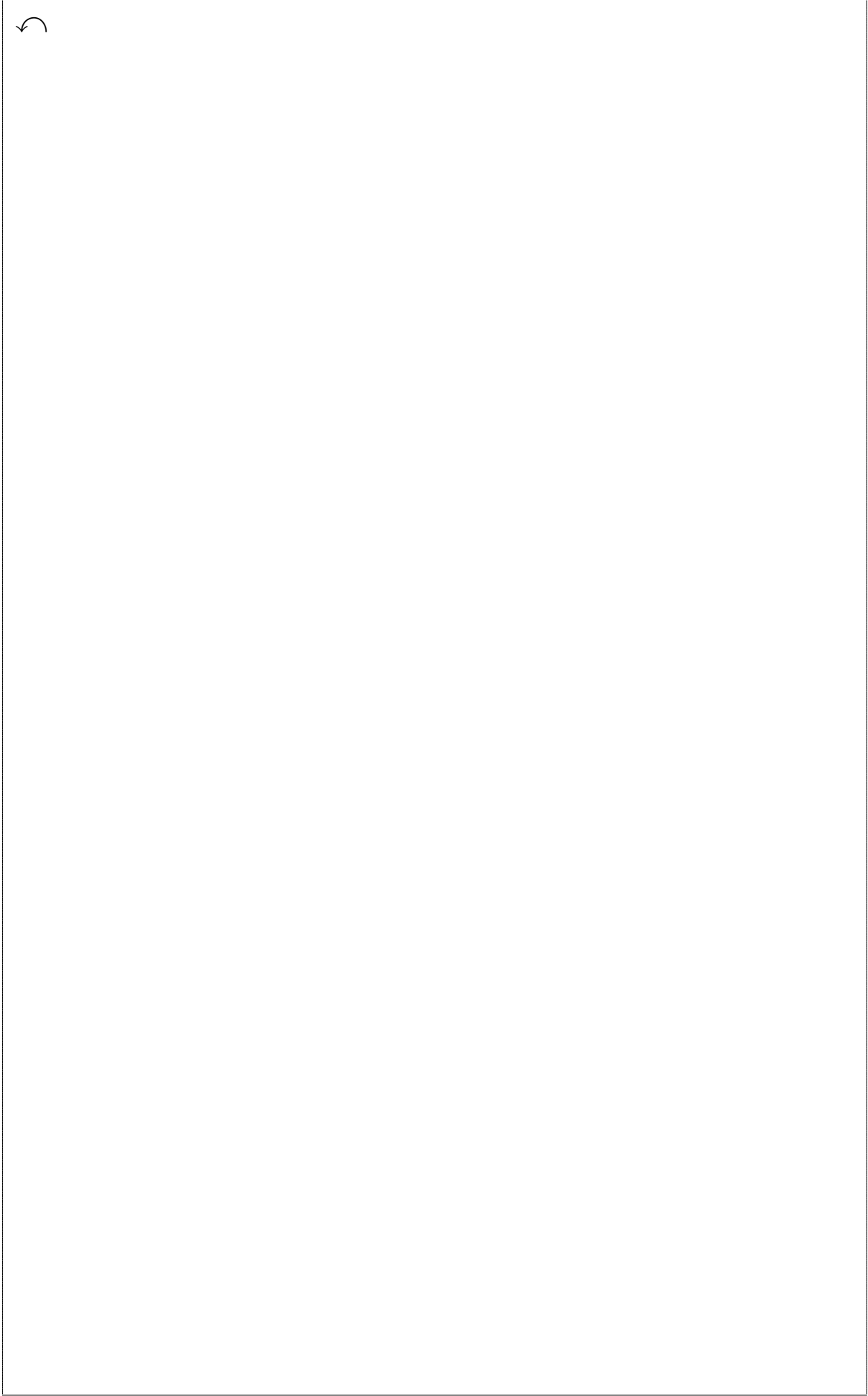
Comme dans une implémentation avec un tableau, le tas est un arbre complet et les éléments de la dernière ligne sont le plus à gauche possible.

- (4 pts) **4.b]** La première étape de l'implémentation est d'être capable de trouver efficacement le dernier nœud du tas (le nœud le plus à droite de la dernière ligne du tas). En fonction de la valeur de `num`, comment savoir si le dernier nœud est la racine, se trouve dans le sous-arbre gauche de la racine, ou se trouve dans le sous-arbre droit de la racine (on suppose que l'on a accès à une fonction `int log(int a)` qui retourne l'entier `h` tel que  $2^h \leq a < 2^{h+1}$ ) ?

- (6 pts) **4.c]** Déduisez-en un algorithme récursif `node* last(node* heap, int num)` qui prend en argument un pointeur `heap` sur la racine d'un tas et le nombre d'éléments `num` de ce tas et retourne un pointeur sur le dernier nœud du tas. Écrivez cet algorithme. Quelle est sa complexité?

- (9 pts) **4.d]** En utilisant la fonction `last` précédente (ou en supposant qu'elle existe si vous n'avez pas réussi à la programmer) écrivez les fonctions `void heap_insert(node* heap, int val)` et `int heap_delete(node* heap)` qui vont respectivement ajouter la valeur `val` dans le tas `heap` et supprimer la racine du tas `heap` en renvoyant la valeur qu'elle contenait. Attention, ces fonctions doivent avoir une complexité en  $O(\log n)$  pour un tas de  $n$  éléments. Pour simplifier, on ne demande pas d'être capable de gérer le cas d'un tas vide. Pensez à bien mettre à jour la valeur du compteur `num` à chaque fois. Pour l'insertion, pensez aussi que la fonction `last` peut être utilisée avec une autre valeur que `num` en argument (par exemple  $(num+1)/2$ ).





**Exercice 5 : Automates pour la reconnaissance de motifs complexes**

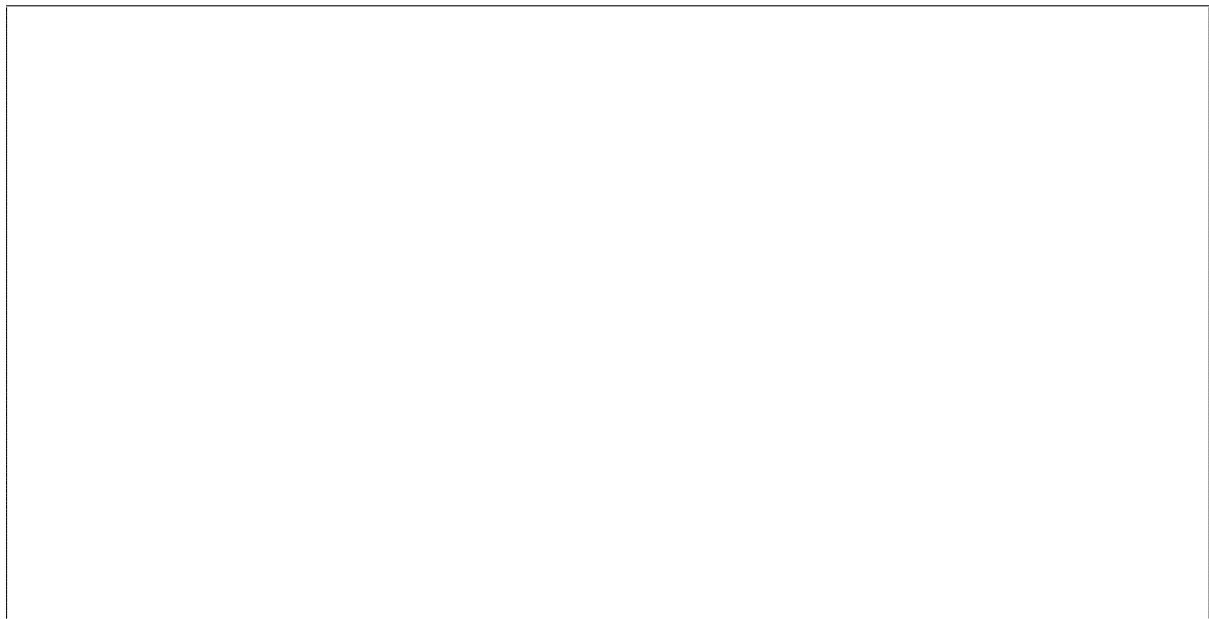
(25 points)

Rappelons que pour un alphabet  $\Sigma$ ,  $\Sigma^*$  représente une suite quelconque (y compris vide) d'éléments de  $\Sigma$ ,  $\Sigma^+$  représente une suite non-vide d'éléments de  $\Sigma$  et  $\Sigma?$  un élément quelconque de  $\Sigma$ . On cherche à reconnaître des motifs de la forme  $m_1\Sigma^*m_2$ ,  $m_1\Sigma^+m_2$  ou  $m_1\Sigma?m_2$  où  $m_1$  et  $m_2$  sont des mots construits à partir de l'alphabet  $\Sigma$ . Le motif  $m_1\Sigma^*m_2$  représente donc le motif  $m_1$  suivi d'une suite quelconque d'éléments de  $\Sigma$  puis du motif  $m_2$ .

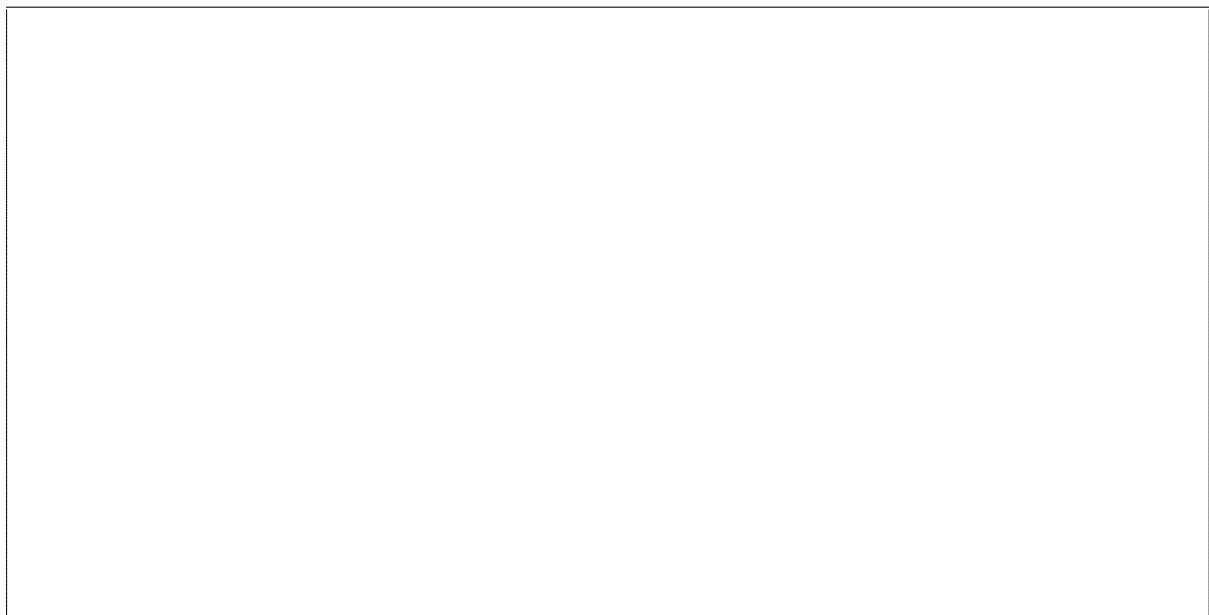
Ici, l'alphabet est l'ensemble des lettres de l'alphabet français de **a** à **z**. Pour alléger les dessins d'automates, une transition identique pour tous les caractères sauf le **a** sera représentée par la notation  $[\hat{a}]$ . De même, toutes les lettres sauf **a** ou **b** sera notée  $[\hat{ab}]$ . Une transition identique pour tous les éléments de l'alphabet  $\Sigma$  sera représentée avec un  $\Sigma$ .

- (4 pts) **5.a]** Dans le cours nous avons vu comment construire des automates reconnaissant un motif  $m$  quelconque, et donc, tous les mots du langage  $\mathcal{L} = \Sigma^*m\Sigma^*$ . Il s'agit donc de construire des automates qui arrivent dans leur état final quand le motif est rencontré et qui restent dans cet état final quel que soit la suite du texte lu.

Construisez les deux automates reconnaissant les motifs **aba** et **pepe**.



- (3 pts) **5.b]** Construisez les 3 automates reconnaissant les langages (attention, il s'agit ici du langage complet, pas d'un motif)  $\Sigma^*$ ,  $\Sigma^+$  et  $\Sigma?$ .



- (6 pts) **5.c]** Vous allez maintenant construire les automates permettant de reconnaître les motifs  $ab\Sigma^*cd$  et  $ab\Sigma^+cd$ . Il suffit pour cela de remarquer que reconnaître l'un de ces motifs consiste à reconnaître un premier motif **ab** puis, dès que ce motif a été reconnu de chercher à reconnaître le motif **cd**. En revanche, comme n'importe quel texte peut se trouver entre les deux motifs, une fois que le premier motif a été trouvé on ne revient plus jamais en arrière (avant le dernier état de l'automate reconnaissant le premier motif). Déduisez-donc des automates reconnaissant **ab** et **cd** les 2 automates reconnaissant les motifs  $ab\Sigma^*cd$  et  $ab\Sigma^+cd$ .

- (5 pts) **5.d]** Vous allez maintenant construire l'automate reconnaissant la motif  $ab\Sigma^?cd$ . Contrairement aux deux motifs précédents, on cherche les deux motifs **ab** et **cd** séparés d'un seul caractère. Donc, une fois que l'on a vu le premier motif, si le motif **cd** n'apparaît pas un caractère après on repart du tout début. Attention, le caractère  $\Sigma^?$  peut aussi être un **a**, modifiant la façon dont on repart au début. Vérifiez en particulier que votre automate reconnaît le texte **ababzcd**. **Toutes les transitions qui ramènent à l'état initial pourront être omises.**

- (7 pts) **5.e]** Construisez l'automate reconnaissant le motif  $ab\Sigma^? \Sigma^?bc$  (les deux motifs sont donc séparés par deux caractères quelconques). Vérifiez que votre automate reconnaît bien  $abzabyzbc$ . **Toutes les transitions qui ramènent à l'état initial pourront être omises.**

Il est interdit de regarder le contenu de ce document avant d'y avoir été invité.

Toute violation de cette interdiction sera sévèrement punie.