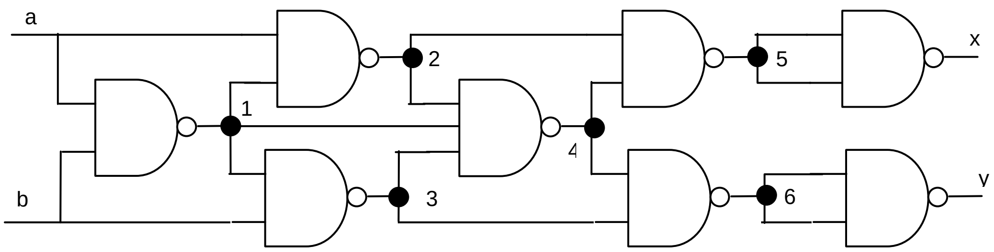
Les exercices sont indépendants. Indiquez en tête de copie le nom du langage que vous utilisez.

I. Logique (15 mn)

I.1. Rappeler la table de vérité de l'opérateur NAND (négation de la conjonction).

Soit le circuit suivant, composé uniquement de portes NAND :



Note : la porte à 3 entrées a en sortie la négation de la conjonction de ses trois entrées.

I.2. Exprimer les expressions des points 1, 2, 3, 4, 5 et 6 en fonction de *a* et *b*.

I.3. En déduire les expressions de *x* et *y* en fonction de *a* et *b*.

II. Théorie des automates (45 mn)

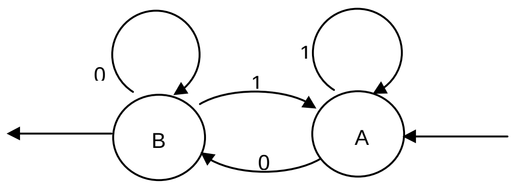


Figure 1

Les automates à deux états sur un alphabet de deux éléments (par exemple, celui de la Figure 1) sont en nombre fini.

II.1. Combien existe-t-il d'automates à deux états sur un alphabet de 2 éléments?

Dans la suite de l'exercice, on notera A l'état initial, B l'état final et les deux éléments de l'alphabet seront 0 et 1.

De plus, on ne considérera que les automates pour lesquels 0 permet la transition de A vers B. (attention, cela n'exclut pas que 1 puisse aussi permettre cette transition).

On s'intéresse aux mots de longueur 4 reconnus par ces automates. Ces mots peuvent représenter les entiers de 0 à 15 codés en base 2. Le mot 1010 représente l'entier 10. Il serait reconnu par l'automate de la figure 1.

II.2. Combien y a-t-il d'automates pour lesquels 1 permet la transition de A vers B ? Représentez-les.

II.3. Pour chacun d'eux, précisez en extension ou en compréhension quels entiers de 0 à 15 seront reconnus comme mots.

III. Les groupes finis (60 mn)

Soient E un ensemble **fini** et T une loi de composition interne de E.
(E, T) peut être représenté par une table de Pythagore.

Si n = card(E), E peut être est mis en bijection avec l'ensemble {1,2, ..., n}. On notera f : $E \rightarrow$ {1,2,...,n} cette bijection et g la bijection inverse.

f étant connue, une matrice nxn peut représenter (E, T) par l'intermédiaire de sa table de Pythagore.

Soit m une matrice représentant (E, T)

III.1. Donner la formule permettant de calculer $m[i, j]$.

III.2. m étant connue, donner la formule permettant de calculer aTb

Dans ce qui suit, on prendra pour E l'ensemble {1,2, ..., n}, f et g seront donc l'identité.

Écrire les fonctions suivantes.

III.3. **`commutative`** **`donnée`** ***`m`*** **`: matrice ; n : entier ;`**
`résultat` ***`r`*** **`: booléen ;`** ***`x, y`*** **`: entier ;`**
si la loi représentée par *m* d'ordre *n* est commutative, met *vrai* dans *r*,
sinon, met *faux* dans *r* et un contre-exemple dans *x* et *y*.

III.4. **`associative`** **`donnée`** ***`m`*** **`: matrice ; n : entier ;`**
`résultat` ***`r`*** **`: booléen ;`** ***`x, y, z`*** **`: entier ;`**
si la loi représentée par *m* d'ordre *n* est associative, met *vrai* dans *r*,
sinon, met *faux* dans *r* et un contre-exemple dans *x*, *y* et *z*.

III.5. **`element_neutre`** **donnée** ***m*** **: matrice ; n : entier ;**
résultat ***e*** **: entier ;**

si la loi représentée par *m* d'ordre *n* possède un élément neutre, met sa valeur dans *e*, sinon, met 0 dans *e*.

III.6. **`element_symetrique_doite`** **donnée** ***m*** **: matrice ;**
n*, *x*, *e **: entier ;**
résultat ***y*** **: entier ;**

(on suppose que la loi représentée par *m* d'ordre *n* possède un élément neutre *e*)
si x possède un symétrique à droite, met dans *y* la valeur du symétrique,
sinon met 0 dans *y*.

III.7. **`element_symetrique_gauche`** **donnée** ***m*** **: matrice ;**
n*, *x*, *e **: entier ;**
résultat ***y*** **: entier ;**

(on suppose que la loi représentée par *m* d'ordre *n* possède un élément neutre *e*)
si x possède un symétrique à gauche, met dans *y* la valeur du symétrique,
sinon met 0 dans *y*.

III.8. **`symetrique`** **donnée** ***m*** **: matrice ;** ***n*, *e*** **: entier ;**
résultat ***r*** **: booléen ;** ***x*** **: entier**

(on suppose que la loi représentée par *m* d'ordre *n* possède un élément neutre *e*)
si tout élément possède un symétrique, met *vrai* dans *r*,
sinon, met *vrai* dans *r*, et un élément sans symétrique dans *x*.

III.9. **`groupe`** **donnée** ***m*** **: matrice ;** ***n*** **: entier ;**
résultat ***r*, *c*** **: booléen ;**

si (*E*, T) représenté par *m* d'ordre *n* est un groupe commutatif, met *vrai* dans *r* et dans *c,*
si (*E*, T) est un groupe non commutatif, met *vrai* dans *r* et *faux* dans *c,*
si (*E*, T) n'est pas un groupe, met *faux* dans *r*.

IV. Permutations (60 mn)

L'objet de cet exercice est l'écriture de programmes permettant d'obtenir les permutations d'une liste d'éléments supposés distincts. Par exemple,
1,2,3   1,3,2   2,1,3   2,3,1   3,1,2   3,2,1

Première approche.

Soit le tableau contenant les n premiers entiers.

a_1	a_2			a_i	a_{i+1}			a_{n-1}	a_n
1	2			i	i+1			n-1	n

L'approche récursive est la suivante :

Supposons que l'on sache générer les permutations sur un tableau de i éléments. Pour générer les permutations d'un tableau de i+1 éléments, on générera tout d'abord toutes les permutations d'un tableau de i éléments, suivies chacune de a_{i+1}. Puis, pour chaque k, avec k<=i, on échangera a_{i+1} avec a_k et on générera les permutations du tableau de i éléments modifié, suivies chacune du nouveau a_{i+1}.
Si i vaut 1, l'affichage de la permutation est évident.

IV.1. Écrire selon le principe ci-dessus la procédure

```
permut   donnée   v : tableau d'entiers ; n : entier ;
```

qui génère et affiche toutes les permutations possibles du tableau v de n entiers.

IV.2. Donner les affichages pour un tableau des 3 entiers 1, 2 et 3.

IV.3. Comment modifier la procédure pour qu'elle affiche les permutations dans l'ordre.

Deuxième approche

La méthode précédente génère toutes les permutations sans qu'il soit possible d'en extraire une particulière. Le but de cette deuxième approche est de générer la permutation suivante d'une permutation existante.

Par exemple, à partir de

5	3	7	4	2	8	6	1

obtenir

5	3	7	4	6	1	2	8

Elle nécessite toutefois de pouvoir disposer d'un ordre sur les éléments à permuter.

Le principe en est le suivant :

- à partir du tableau actuel, repérer l'indice de début du dernier sous-tableau strictement décroissant (éventuellement réduit au dernier élément).

5	3	7	4	2	8	6	1
1	2	3	4	5	6	7	8

Si tout le tableau est décroissant, la génération des permutations est terminée.

- renverser ce sous-tableau

5	3	7	4	2	1	6	8
1	2	3	4	5	6	7	8

- échanger l'élément précédent ce sous-tableau avec le premier élément du sous-tableau qui lui est strictement supérieur

5	3	7	4	6	1	2	8
1	2	3	4	5	6	7	8

IV.4. Justifier la validité de cette approche.

IV.5. Écrire la fonction

dernier_sstab_decroissant **donnée** ***t*** **: tableau d'entiers ;**
n **: entier ;**
résultat ***r*** **: entier**

qui retourne l'indice du dernier sous-tableau décroissant de *t* (tableau de *n* entiers)

IV.6. Écrire la fonction

renverse **donnée-résultat** ***t*** **: tableau d'entiers ;**
donnée ***a*, *b*** **: entier ;**

qui renverse le sous-tableau de *t* indicé de *a* à *b*. (*t*[*a*] est échangé avec *t*[*b*], *t*[*a*+1] est échangé avec *t*[*b*-1], etc.)

IV.7. Écrire la fonction

indice_plus_grand **donnée** ***t*** **: tableau d'entiers ;** ***n*, *a*** **: entier ;**

qui retourne l'indice du premier élément du tableau situé entre l'indice *a*+1 et *n* strictement plus grand que *t*[*a*]

IV.8. Écrire la fonction

permutation_suivante **donnée-résultat** ***t*** **: tableau d'entiers ;**
donnée ***n*** **: entier ;**

qui transforme le tableau *t* de *n* entiers de manière à lui faire contenir la permutation suivante.