

SERIE DE DEVOIRS SURVEILLES EN INFORMATIQUE

Proposée par Pr. D. El Ghanami

Ecole Mohammadia d'Ingénieurs

Mai 2011

Achat de marchandise dans un magasin

- ✓ On attachera une importance à la concision, à la clarté, et à la précision de la rédaction en langage C.
- ✓ La fonction prédéfinie autorisée est *printf*.
- ✓ La liste ne doit pas être transformée en tableau pour traiter les questions de la partie 2.
- ✓ a et b deux entiers positifs, a/b donne le quotient de la division et $a\%b$ donne son reste.

On veut aider un client dans l'achat de produits dans un magasin. Ce magasin possède un catalogue de N produits différents qu'il vend représenté par un tableau T de produits.

T est déclaré dans la fonction *main()*. N est une constante globale.

Chaque produit a un numéro *no* et un prix de vente *prix* en dirhams (des entiers positifs). Chaque produit a un prix unique dans le catalogue et disponible en nombre d'exemplaires suffisant dans le magasin.

Le type produit proposé est :

```
struct produit {
    int no
    int prix ;
};
```

avec la déclaration de synonyme : `typedef struct produit prod ;`

Exemple de catalogue T de 10 produits différents de numéros : 1 à 10 et des prix : 720dh, ..., 480dh

1	2	3	4	5	...	10
720	204	600	120	500	...	480

Un client se présente dans le magasin avec un montant d'argent m en dirhams et un camion de capacité de stockage c pour acheter et transporter des produits.

m et c des entiers positifs déclarés dans la fonction *main()*.

Chaque produit est mis dans un carton pour le transporter. Tous les cartons ont le même volume v (entier constant). Un carton ne peut contenir qu'un seul exemplaire de produit.

Pour rentabiliser son déplacement chez le magasin, le client décide de dépenser le maximum d'argent sans oublier la capacité du camion.

Exemple : l'achat d'un produit de 20dh est mieux que deux produits de 10dh. L'achat de 3 produits $n^{\circ}3$ et de 4 produits $n^{\circ}5$ coûtera 3800dh et occupera un espace camion de $7v$.

Le client simule des calculs pour décider quels produits acheter et en combien d'exemplaires.

On veut l'aider dans ses calculs en lui proposant des fonctions développées selon sa logique.

Partie 1

Sachant que le volume d'un carton est v et le prix du produit le moins cher est pm et sans programmer, que pouvez-vous dire :

1. Du montant d'argent m qui restera à la fin d'achat si la capacité du camion c est illimitée ?
2. De l'espace c qui restera dans le camion à la fin d'achat si m est très grand ?

Au début, le client cherche à acheter le maximum de produits avec son argent.

3. Ecrire la fonction *prodSelect(T)* qui retourne *px* le produit qui peut être acheté le plus.
4. Ecrire la fonction *nbProd(m, c, px)* qui donne le nombre d'exemplaires achetés de ce produit.

Cependant, ce calcul remplit le camion mais le client lui restera un montant d'argent important non dépensé à cause de la capacité de stockage *c* qui est limitée.

De la même manière, il calcule le minimum de produits qu'il peut acheter.

Ainsi, il constate qu'il lui reste encore de l'espace dans le camion et moins d'argent que précédemment. Par conséquent, il décide d'adopter ce calcul en continuant l'achat, à chaque fois, du produit le plus cher possible par l'argent qui reste à chaque fois.

Pour faciliter la sélection des produits, il les classe par ordre décroissant de prix et cela selon l'approche suivante : il cherche le produit le plus cher et l'insert à la 1^{ère} place, puis reprend cette démarche pour les autres produits jusqu'à obtenir un catalogue trié selon les prix.

Exemple de prix de produits :

1	2	3	4	5	6	7
120	204	<u>600</u>	123	500	456	480

Quelques étapes de tri :

3	1	2	4	5	6	7
<u>600</u>	120	204	123	<u>500</u>	456	480

3	5	1	2	4	6	7
<u>600</u>	<u>500</u>	120	204	123	456	<u>480</u>

3	5	7	1	2	4	6
<u>600</u>	<u>500</u>	<u>480</u>	120	204	123	<u>456</u>

3	5	7	6	1	2	4
<u>600</u>	<u>500</u>	<u>480</u>	<u>456</u>	120	<u>204</u>	123

3	5	7	6	2	1	4
<u>600</u>	<u>500</u>	<u>480</u>	<u>456</u>	<u>204</u>	120	<u>123</u>

5. Ecrire la fonction *classCat(T)* qui classe les produits du catalogue selon le principe ci-dessus.

Après tri, le client commence l'achat et le stockage de produits dans le camion. Lors de cet achat, le client sent qu'il répète, après chaque sélection de produit, la même opération comme s'il vient d'arriver dans le magasin avec un nouveau montant d'argent *m* inférieur et une nouvelle capacité du camion *c* inférieure.

6. Ecrire la fonction récursive *restArgent(T, m, c, ...)* qui donne l'argent qui reste à la fin d'achat.

N.B. Les ... en paramètre de la fonction signifient qu'il peut y avoir d'autres paramètres à définir.

7. Cette fonction peut-elle être écrite, par simple transformation, en itérative ? (expliquer).
8. Ecrire la fonction itérative *espLibre(T, m, c)* qui donne l'espace restant dans le camion à la fin d'achat.

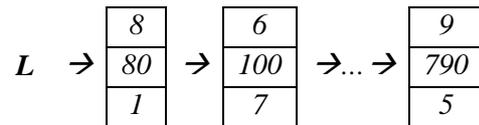
Partie 2

Lors d'achat de produits, le client enregistre les informations relatives aux différents produits achetés dans une liste chaînée L (le produit pr et le nombre d'exemplaires achetés de chaque produit ne). Un produit sélectionné ne doit figurer qu'une seule fois dans la liste.

Le type liste proposé est :

```
typedef struct tliste{
    prod pr ;
    int ne ;
    struct tliste *s ;
} liste ;
```

Exemple de liste :



On suppose que cette liste L est déclaré dans la fonction $main()$ par l'instruction :

```
liste *L = NULL ;
```

L'opération de sélection des produits du catalogue suit le principe cité en partie1 et l'enregistrement dans la liste L consiste à empiler les informations des produits sélectionnés dans la liste selon le principe le dernier produit acheté est le premier de la liste.

9. Ecrire la fonction $listeProd(L, T, m, c)$ qui donne la liste des produits achetés et leur nombre.
10. Ecrire la fonction $totalDep(L)$ qui donne le montant global dépensé lors de l'achat de produits.
11. Ecrire la fonction récursive $affiche(L)$ qui affiche les prix des produits du plus au moins cher.

Des points dans le plan

- ✓ On attachera une importance à la concision, à la clarté, et à la précision de la rédaction en langage C.
- ✓ Il est conseillé de lire la totalité de l'énoncé avant de commencer.
- ✓ La fonction autorisée est *sqrt(x)* qui retourne la racine carrée du réel x .
- ✓ Une fonction développée dans une question peut être utilisée même si elle est incomplète.

Dans ce problème, on considère deux tableaux de réels T_x et T_y de taille N représentant les positions de N points différents dans le plan. T_x contient les abscisses x et T_y les ordonnées y des N points.

T_x et T_y sont des variables globales et N est une constante globale.

x et y sont des réels. Ainsi, le plan est divisé en deux demi plan : demi plan sup ($y \geq 0$) et demi plan inf ($y < 0$).

Exemple de 10 points :

i	0	1	2	3	4	5	6	7	8	9
T_x	-0,7	-0,1	-0,5	0,5	1	1	-0,7	-1	0	0
T_y	1,2	-1	-1,3	1,3	-1	1	0,6	1	-1,4	0

Le 1^{er} point ($i = 0$) de coordonnée $x = -0,7$ et $y = 1,2$ se trouve dans le demi plan sup.

Le 2^{ème} point de coordonnée $x = -0,1$ et $y = -1$ se trouve dans le demi plan inf.

I- On cherche à calculer la distance entre deux points quelconques du plan.

1. Ecrire la fonction *float car(float x)* qui donne le carré du réel x .
2. Ecrire la fonction *float dist1(int i, int j)* qui donne d_{ij} la distance entre les points d'indices i et j .
3. Ecrire la fonction *float dist2()* qui donne d_{max} la distance entre les 2 points les plus éloignés l'un de l'autre.
4. Ecrire la fonction *float dist3(int i)* qui donne d_i la distance entre le point i et l'origine $(0,0)$.

On veut calculer r_{max} le rayon du cercle, autour de l'origine, formé par les points du plan les plus loin de l'origine et le nombre K de points appartenant à ce cercle. K et r_{max} sont déclarés globaux.

5. Ecrire la fonction *void cercleMax()* qui calcule r_{max} et K (donner une solution en un seul parcours de T_x et T_y).

II- On suppose que les coordonnées des K points formant ce cercle sont stockés dans une matrice de réels M de taille $2 \times K$ déclarée globale. La 1^{ère} ligne contient les abscisses x et la 2^{ème} ligne contient les ordonnées y .

De l'exemple précédent, M est :

i	0	1	2	3	4	5	6
x	-0,7	-0,5	0,5	1	1	-1	0
y	1,2	-1,3	1,3	-1	1	1	-1,4

On veut trier les points de la matrice M dans le sens de rotation des aiguilles de la montre en commençant par le demi plan sup.

Pour cela, on propose d'abord le rangement des éléments de M de sorte à avoir 2 zones dans M , les points du demi plan sup au début de M suivi des points du demi plan inf.

Le rangement ne doit pas se baser sur le tri des points, qui nécessite plus d'opérations inutile, mais sur la permutation, deux à deux, de points qui nécessitent de changer de zone (voir exemple ci-dessous).

Les étapes de rangement de M :

i	0	1	2	3	4	5	6
x	-0,7	-0,5	0,5	1	1	-1	0
y	1,2	-1,3	1,3	-1	1	1	-1,4

i	0	1	2	3	4	5	6
x	-0,7	-1	0,5	1	1	-0,5	0
y	1,2	1	1,3	-1	1	-1,3	-1,4

i	0	1	2	4	3	5	6
x	-0,7	-1	0,5	1	1	-0,5	0
y	1,2	1	1,3	1	-1	-1,3	-1,4

6. Ecrire la fonction `void permute(int i, int j)` qui permute deux points de M d'indices i et j .
7. Ecrire la fonction `void range()` qui permet de réaliser le rangement de M selon le principe précédent.
8. Ecrire la fonction `int indic()` qui donne *inf* indice du 1^{er} point du demi plan inf après rangement de M .

Le tri de M donne :

i	0	1	2	3	4	5	6
x	-1	-0,7	0,5	1	1	0	-0,5
y	1	1,2	1,3	1	-1	-1,4	-1,3

9. Ecrire la fonction `void triCercle()` qui trie la matrice M dans l'ordre cité ci-dessus.

III- Après avoir trié la matrice M , on cherche à vérifier si un point P de coordonnées $x1$ et $y1$ appartient à ce cercle. Le fait que la matrice soit triée accélère la recherche qui se fait en deux étapes :

- La désignation d'abord dans M de la zone correspondant au demi plan où la recherche sera effectuée.
- Dans la zone du demi plan concerné par la recherche, on compare $x1$ et $y1$ avec x et y du point se trouvant à l'indice milieu im de cette zone, si ils sont égaux alors P appartient au cercle sinon on recommence la recherche dans la zone à gauche ou à droite de im selon le résultat de la comparaison cela jusqu'à la fin de la recherche.

En supposant que la recherche se fait dans le demi plan sup

10. Ecrire la fonction récursive `int rechSup(float x1, float y1, int ideb, int ifin)` qui recherche si le point P appartient à ce demi plan ou pas et cela en respectant le principe de recherche cité ci-dessus.

En supposant que la recherche se fait dans le demi plan inf

11. Ecrire la fonction itérative `int rechInf(float x1, float y1, int ideb, int ifin)` qui recherche si le point P appartient à ce demi plan ou pas et cela en respectant le principe de recherche cité ci-dessus.
12. Ecrire la fonction `recherche(float x1, float y1)` qui vérifie si le point P appartient à M ou pas.

Compression et Brouillage d'images

On veut construire un outil de compression et de brouillage d'images (appelé encodeur) ainsi que l'outil capable de les décoder et de les décompresser (appelé décodeur).

I- Représentation d'une image par une matrice de caractères.

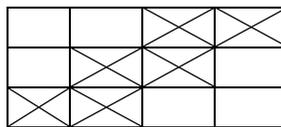
Pour simplifier le problème, nous considérons des images en noir et blanc. Une image est décrite ligne par ligne par des points (pixels) représentés par des caractères (caractère 'b' pour les points blancs et caractère 'n' pour les points noirs). Ces lignes ont le même nombre de points.

Exemple : La matrice de caractères {"nbn", "bnbb", "bbbb"} représente l'image 3x4 suivante :

'n'	'b'	'n'	'n'	'\0'
'b'	'n'	'b'	'b'	'\0'
'b'	'b'	'b'	'b'	'\0'

Le caractère '\0' ne fait pas partie de l'image mais indique la fin d'une chaîne de caractères (fin d'une ligne).

1- Donnez la matrice de caractères correspondant à l'image suivante :



2- Dessinez l'image correspondant à la matrice suivante : {"bnb", "nnn", "nbn"}

II- Représentation d'une image par une séquence d'entiers.

L'objectif de la compression d'image est de représenter une image par une séquence d'entiers plus courte que la matrice de caractères 'n' et 'b'.

Pour compresser une image, on suit les étapes suivantes :

- On forme d'abord une matrice d'entiers à partir de l'image où à chaque ligne de l'image correspond une ligne de la matrice d'entiers délimitée par l'entier -1.

La 1^{ère} valeur d'une ligne indique le nombre de points blancs du début de la ligne (le nombre vaut 0 si la ligne ne commence pas par des points blancs) ;

La 2nd valeur indique le nombre de points noirs qui suivent, la 3^{ème} valeur indique le nombre de points blancs qui suivent, et ainsi de suite jusqu'à la fin de la ligne.

- Puis, on transforme la matrice d'entiers en séquence d'entiers où -1 indique la fin de chaque ligne et la suite -1 -1 indique la fin de l'image (fin de la séquence d'entiers).

Les valeurs de la séquence sont stockées dans un tableau.

De l'exemple précédent, on forme la matrice d'entiers suivante :

0	1	1	2	-1
1	1	2	-1	
4	-1			

Ainsi, la séquence est :

0	1	1	2	-1	1	1	2	-1	4	-1	-1
---	---	---	---	----	---	---	---	----	---	----	----

3- Donnez la matrice de caractères de l'image correspondant à la séquence d'entiers :

$\{0, 2, 1, -1, 2, 2, 2, -1, 0, 1, 2, -1, -1\}$

4- Donnez la séquence d'entiers correspondant à l'image {"bnb", "nnn", "nbn"}

5- Définir les constantes globales suivantes : NL (nombre de lignes de l'image), NC (nombre de colonnes de l'image) et MAX (taille maximale du tableau d'entiers contenant la séquence).

6- Pourquoi MAX doit être égal à $NL*(NC+2)$?

7- Ecrire la fonction `void afficheSeq(int seq[])` qui affiche les entiers de la séquence séparés par des virgules jusqu'au marqueur de fin de séquence.

III- Compression d'images

8- Ecrire la fonction `void compresseImage(char image[][NC+1], int seq[])` qui permet de compresser une image de taille $NL \times NC$.

Voici quelques fonctions à développer qui peuvent vous aider.

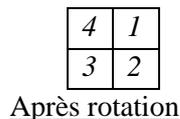
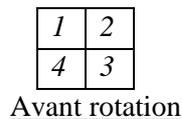
9- La fonction `void compresseL(char limage[], int lmat[])` qui permet de compresser un tableau de caractères.

10- La fonction `compresserMat (char image[][NC+1], int matEnt[][NC+2])` qui permet de compresser une image dans une matrice d'entiers.

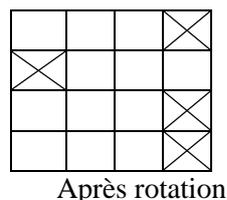
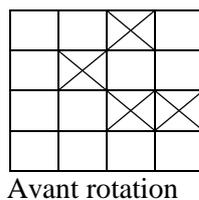
11- La fonction `void compresseSeq(int matEnt[][NC+2], int seq[])` qui forme la séquence d'entiers compressée à partir d'une matrice d'entiers.

IV- Brouillage et décodage d'une image

Le principe du brouillage D.E.S. consiste à faire une rotation de 90° , dans le sens des aiguilles de montre, d'un carré de quatre points comme indiqué ci-dessous :



Pour brouiller une image on applique cette permutation à chacun des carrés de quatre points de l'image comme indiqué sur l'exemple suivant :



On suppose que le nombre de lignes et de colonnes d'une image est toujours pair.

12- Ecrire la fonction `void brouillerImage (char image[][NC+1])` qui permet de brouiller une image suivant le sens indiqué précédemment.

Voici une procédure, à développer, qui peut vous aider.

13- La fonction *void brouillerCarre (char image[][NC+1], int l, int c)* permet de brouiller un carré d'image d'indice *l* et *c*.

14- Ecrire une procédure qui permet de brouiller et de compresser une image noir et blanc.

V- Décodage et décompression d'une image

15- Proposer la fonction *void decompresserImage (int seq[], char image[][NC+1])* qui réalise le décodage et la décompression d'une image brouillée et compressée. Son prototype est :

Indication. En brouillant une image déjà brouillée, obtient-on une image encore brouillée ?

Cryptage d'image

On cherche à crypter une image numérique en couleur de taille 100×100 pixels. Un pixel (ou point) est une couleur et une intensité lumineuse.

Dans ce problème, on ne considérera que les couleurs des pixels. Une couleur est un entier codé sur 8 bits, soit 256 couleurs différentes **représentées par des entiers compris entre 0 et 255**.

Ainsi, les couleurs seront numérotées de 0 à 255 avec 0 représentant le blanc et 255 représentant le noir.

Exemple : L'image de la figure1 de 3×3 pixels est représentée par la matrice M de taille 3×3 (figure2)

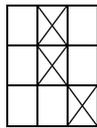


Figure1

0	255	0
0	255	0
0	0	255

Figure2

0	255	0	0	255	0	0	0	255
---	-----	---	---	-----	---	---	---	-----

Figure3

Soit une image de $N \times N$ pixels ($N = 100$) et une fonction *remplirImage(M)* qui permet, en l'appelant, de remplir une matrice M d'entiers de taille $N \times N$ à partir cette image.

Pour crypter cette image, on utilise le code de César. Le principe est de modifier les couleurs de l'image par rapport à une clé qui est aussi une couleur.

La clé K (représentée par un entier entre 0 et 255) donne le décalage d entre la couleur de la clé K et le blanc de numéro 0.

Par exemple, en décalant les couleurs de I position, le blanc devient la couleur numéro I , la couleur numéro $0 < i < 255$ devient la couleur numéro $i+I$, le noir devient le blanc.

1. Que donne le codage de l'image précédente en utilisant un décalage de 10 ?
2. Ecrire un programme (fonction main) qui permet de :

- 2.1. Déclarer et initialiser une matrice M à partir d'une image non cryptée.
- 2.2. Lire et contrôler un nombre entier K positif représentant la clé ($0 \leq K < 256$)
- 2.3. Crypter l'image M .

Pour réaliser le décryptage, il faut connaître la valeur du décalage d . Une manière de la déterminer est d'essayer de deviner cette valeur. L'approche proposée est de chercher la couleur dominante dans l'image cryptée (on suppose qu'elle est unique). En effet, la couleur dominante dans une image non cryptée est la couleur X de numéro 100.

3. Ecrire un programme (main) qui permet de :

- 3.1. Déclarer et initialiser une matrice M à partir d'une image cryptée.
- 3.2. Chercher la couleur dominante dans cette image.
- 3.3. Déduire la clé du cryptage pour que la couleur X soit la plus fréquente dans l'image décrypté.
- 3.4. Décrypter l'image M .

Etude d'une carte géographique

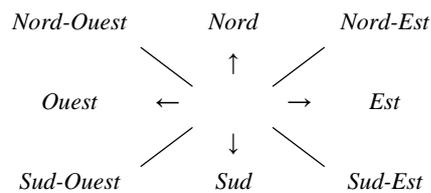
- ✓ On attachera une importance à la concision, à la clarté, et à la précision de la rédaction en langage C.
- ✓ Aucune fonction prédéfinie n'est autorisée.

Soit une carte géographique d'une région sous forme d'une grille T (matrice de taille $N \times M$) représentant les coordonnées de $N \times M$ points géographiques (abscisses, ordonnées et altitudes).

La carte T est orientée (Nord, Sud, Est et Ouest) et N et M sont des constantes globales.

Exemple : T de taille 6×9 :

$i \backslash j$	0	1	2	3	4	5	6	7	8
0	6	4	4	4	4	2	1	0	2
1	5	4	3	2	1	5	5	4	4
2	5	5	5	1	0	1	5	6	6
3	4	6	7	6	7	6	7	9	8
4	2	3	8	10	8	8	6	6	4
5	1	4	5	8	8	8	4	4	5



Les points de T sont identifiés par leur position h dans T ($1 \leq h \leq N \times M$). h est ordonné du Nord-Ouest ($i=0, j=0$) vers le Sud-Est ($i=N-1, j=M-1$) en passant ligne par ligne.

La relation entre h et les indices ligne i et colonne j de T est $h=i \times M+j+1$.

Ainsi, $i=(h-1)/M$ et $j=(h-1)\%M$.

Le type de T est :

```
typedef struct {
    int a ;
    char c ;
} elt ;
```

a est un entier positif représentant l'altitude d'un point de T .

c sera utilisé pour marquer un point de T . Sa valeur par défaut est ' '. (espace)

Un point de T est de coordonnées $(i, j, T[i][j].a)$, i est son abscisse, j son ordonnée et $T[i][j].a$ son altitude.

Exemple : les points 1, 9 et 10 ont respectivement les coordonnées $(0, 0, 6)$, $(0, 8, 2)$ et $(1, 0, 5)$.

Les points de la matrice T sont de deux types :

- des points bords correspondant aux indices $i = 0, i = N-1, j=0$ ou $j = M-1$.
- des points centres correspondant aux indices $1 \leq i < N-1$ et $1 \leq j < M-1$.

Un point centre ne peut pas être un point bord et inversement.

Dans ce problème, on veut étudier les descentes de la carte T à partir de son sommet le plus haut. Pour cela, on développera des fonctions permettant de tracer les descentes et étudier une d'elle.

Définition : Une descente de T est le passage d'un point d'altitude $a1$ à un de ses points voisins d'altitude $a2$ tel que $a1 > a2$. La descente continue jusqu'à ne plus pouvoir descendre.

Partie 1 : Tracé de descentes

I- On cherche d'abord à tracer une des descentes de T . Le tracer consiste à marquer la variable c des points qui composent la descente par la lettre 'd' ($T[i][j].c = 'd'$).

Pour cela, il faut marquer le sommet le plus haut de T puis marquer à chaque fois un point descente tant qu'il existe ou atteindre un point bord de T .

1. Ecrire la fonction $f(i, j)$ qui retourne h la position d'un point de T à partir de ses indices i et j .
2. Ecrire la fonction $abs(x)$ qui retourne la valeur absolue de l'entier x .
3. Ecrire la fonction $bord(i, j)$ qui vérifie si le point d'indices i et j est un point bord de T ou pas.

Un point centre est un sommet si $T[i][j].a > T[k][l].a$ pour $i-1 \leq k \leq i+1$ et $j-1 \leq l \leq j+1$ avec $k \neq i$ et $l \neq j$.

Exemple : Dans T , le point de coordonnée $(3, 7, 9)$ est un sommet.

4. Ecrire la fonction $sommet(T, i, j)$ qui vérifie si le point centre d'indices i et j est un sommet ou pas.

Un point descente est un point voisin d'un point centre d'indices i et j permettant une descente. Dans le cas de plusieurs points descentes alors on retourne le 1^{er} point, dans l'ordre de h présenté précédemment.

Exemple : Dans T , la 1^{ère} descente à partir du point de coordonnées $(2, 7, 6)$ est le point $(1, 6, 3)$.

5. Ecrire la fonction $point(T, i, j)$ qui retourne h la position du 1^{er} point descente s'il existe sinon 0.

II- On trace ici la descente à partir du sommet le plus haut en passant à chaque fois par le 1^{er} point descente.

Exemple : La figure suivante présente les tracés de la 1^{ère} descente de T à partir du point $(4, 3, 10)$ et celle à partir du point $(3, 7, 8)$.

$i \backslash j$	0	1	2	3	4	5	6	7	8
0	6	4	4	4	4	2	1	0	2
1	5	4	3	2	1	5	5	4	4
2	5	5	5	1	0	1	5	6	6
3	4	6	7	6	7	6	7	9	8
4	2	3	8	10	8	8	6	6	4
5	1	4	5	8	8	8	4	4	5

6. Ecrire la fonction $sommetMax(T)$ qui retourne h la position du sommet le plus haut de T s'il existe.
7. Ecrire la fonction $descente(T, i, j)$ qui marque le point sommet le plus haut d'indices i et j et tous les points de sa 1^{ère} descente par la lettre 'd'.

Sachant que le tracé d'une descente composée de k points est le marquage du 1^{er} point et le tracé d'une descente de $k - 1$ points.

8. Ecrire une version récursive de la fonction $descente(T, i, j)$.

On veut tracer toutes les descentes à partir du sommet le plus haut de la carte. Le tracé consiste à marquer le point sommet et toutes ses descentes.

Remarque importante : Chaque point de ce tracé peut être vu avec les descentes qui le traversent comme un tracé de descentes.

- Ecrire la fonction récursive $lesDescentes(T, i, j)$ qui marque le point d'indices i et j et toutes ses descentes par la lettre 'd'.

Partie 2 : Etude d'une descente

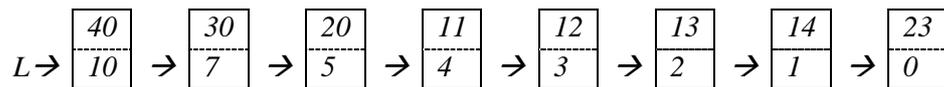
Les points d'une descente partant du sommet le plus haut de T forment la liste chaînée L . Le point sommet de T se trouve au début de la liste L .

Le type de L est :

```
typedef struct tliste{
    int h, a ;
    struct tliste *s ;
} liste ;
```

h est la position du point dans T et a son altitude.

Exemple de L :



III- Dans cette partie, on veut étudier les caractéristiques de la descente L afin de l'exploiter pour des activités sportives (ex. ski).

Quelques caractéristiques de T : La distance entre le point 30 et le point 20 est 2. La longueur de L est 10.

- Sachant que la distance entre 2 points voisins de L est la différence entre leur altitude, écrire la fonction $dist2(p)$ qui retourne la distance d entre le point $*p$ et son suivant s'il existe sinon 0.
- Ecrire la fonction $longueur(L)$ qui retourne la longueur de la descente L .
- Ecrire la fonction $moyen(L)$ qui retourne la moyenne des distances d entre les points successifs de L .

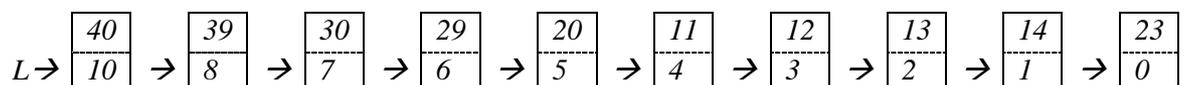
Sachant que la longueur d'une descente à partir du point p est égale à la somme de $dist2(p)$ et la longueur de la descente à partir de son suivant.

- Ecrire une version récursive de la fonction $longueur$.

IV- On désire allonger la descente L tant que c'est possible afin d'affaiblir sa force de descente. Pour cela, on cherche à augmenter le nombre de points de L tout en gardant sa longueur. Pour cela, on adopte la démarche suivante :

Soit le point $*p$ de L et son suivant (le point suivant est la descente de $*p$), alors on cherche un point h de T intermédiaire entre eux de sorte qu'il maintient la descente. h peut être un point quelconque (bord ou centre). S'il y'a plusieurs points intermédiaires h , alors on prend le 1^{er} point dans l'ordre de h .

Exemple : L ajustée:



La descente représentée dans T (figure à gauche). La figure à droite représente le sens de la descente.

ij	0	1	2	3	4	5	6	7	8
0	6	4	4	4	4	2	1	0	2
1	5	4	3	2	1	5	5	4	4
		'd'	'd'	'd'	'd'				
2	5	5	5	1	0	1	5	6	6
		'd'			'd'				
3	4	6	7	6	7	6	7	9	8
		'd'	'd'						
4	2	3	8	10	8	8	6	6	4
			'd'	'd'					
5	1	4	5	8	8	8	4	4	5

ij	0	1	2	3	4	5	6	7	8
0									
1		→	→	→	↓				
2		↑			X				
3		↑	←						
4			↑	X					
5									

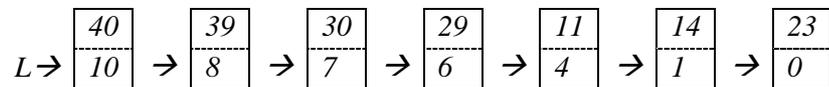
14. Ecrire la fonction $interim(T, p)$ qui retourne le point intermédiaire h s'il existe sinon 0.

15. Ecrire la fonction $ajuster(T, L)$ qui ajuste tant que c'est possible le tracé de la descente L .

III- Les directions de la carte T sont Est/Ouest, Nord/Sud, Nord-Ouest/Sud-Est et Nord-Est/Sud-Ouest (voir l'orientation de la carte T).

Dans cette partie, on veut factoriser la descente L . La factorisation consiste à garder dans L que les points présentant un changement de direction. Pour cela, on supprime de chaque séquence de points continue de L se trouvant dans la même direction tous les points milieu et on ne garde que les deux points d'extrémités.

Exemple : L précédente factorisée



16. Ecrire la fonction $factor(L)$ qui factorise la liste L .

Organisation d'élections

- ✓ On attachera une importance à la concision, à la clarté, et à la précision de la rédaction en langage C.
- ✓ La fonction prédéfinie autorisée est *printf*.

Dans un pays, N parties politiques ont présenté leur liste de candidats aux élections (listes électorales numérotées de 1 à N). Ces élections ont vu la participation de M votants, chacun inscrit un numéro de liste dans un bulletin de vote saisi dans un tableau de votes V de taille M . $V[i] = j$ signifie que le vote i ($0 \leq i < M$) est partie pour la liste j ($1 \leq j \leq N$). Un vote blanc ou annulé n'est pas considéré vote valide et $V[i]=0$.

N et M sont des constantes globales.

Exemple de 25 votants sur 6 listes :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
V	3	3	0	2	5	2	3	0	2	1	3	5	5	0	2	3	4	0	0	6	5	5	5	6	6

Les résultats des listes sont stockées dans un tableau P de taille $N+1$ où $P[i]=j$ signifie que le nombre de votes valides obtenus par la liste i est j .

$P[0]$ contient le total des votes blancs ou annulés.

Exemple : Les résultats de l'exemple précédent :

i	0	1	2	3	4	5	6
P	5	1	4	5	1	6	3

Le problème consiste à aider la commission d'organisation des élections à appliquer les règles électorales pour décompter les résultats, nommer le gagnant majoritaire ou prévoir la coalition de listes idéale.

I- Une liste obtenant plus de 50% des voix valides est considérée majoritaire et désignée vainqueur des élections. De l'exemple précédent, la liste qui obtient au moins 11 voix valides gagnera les élections.

1. Ecrire la fonction *init(T, L)* qui initialise le tableau d'entiers T de taille L à 0.
2. Ecrire la fonction *scores1(V, P)* qui remplit P tableau des résultats des listes et les votes blancs.
3. Ecrire la fonction *nbVoix(P)* qui retourne *nvv* le nombre de voix valides des élections.
4. Ecrire la fonction *gagnant1(P)* qui retourne le gagnant majoritaire s'il existe sinon -1.

Après décompte des voix, on élimine de V les voix non valides en préservant l'ordre des votes.

Exemple : V précédent devient :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...	24
V	3	3	2	5	2	3	2	1	3	5	5	2	3	4	6	5	5	5	6	6

5. Ecrire la fonction *purger(V)* qui réalise l'élimination des voix non valides de V .
6. Si V est trié (ex. tri décroissant), écrire la fonction *gagnant2(V, nvv)* qui retourne le gagnant majoritaire.

II- En cas d'absence de vainqueur majoritaire, un 2nd décomptage est effectué pour le désigner. Sa raison, c'est que la participation de nombreuses listes empêche une des listes principales d'une victoire majoritaire.

Les listes qui ont moins de 10% des voix sont éliminées (remise de leur score à 0). Les voix de ces listes *nva* sont redistribuées totalem sur les listes restantes, en proportion de leur score, grâce à la fonction *f* supposée existée.

Si le score d'une liste est *j* alors *j* est augmenté de $f(j, nva, nvv, nlr)$ où *nlr* est le nombre de listes restantes.

Exemple : Le 2nd décomptage de l'exemple précédent donne :

<i>i</i>	0	1	2	3	4	5	6
<i>P</i>	5	0	4	6	0	7	3

7. Ecrire la fonction *scores2(P)* qui mettra à jour les scores des listes après 2nd décomptage.

III- Les listes restantes sont stockées dans un tableau *T* de taille *nlr* contenant leur numéro *nl*, leur score *sl*, leur courant politique *cl* et la liste des noms des candidats *L*.

La valeur de *cl* appartient aux 3 courants politiques classiques : gauche, droite et centre représentés respectivement par les 3 lettres 'g', 'd' et 'c'.

Le type de *T* est :

```
struct element{
    int nl, sl;
    char cl ;
    listeC *L ;
};
```

Avec le synonyme suivant : `typedef struct element elt ;`

Le type *listeC* est :

```
typedef struct listeCandidat{
    char nom[20];
    struct listeCandidat *s ;
} listeC ;
```

Exemple : *T* correspondant à l'exemple précédent :

<i>i</i>	<i>nl</i>	<i>sl</i>	<i>cl</i>	<i>L</i>	
0	2	4	'g'		→...
1	3	6	'c'		→...
2	5	7	'g'		→...
3	6	3	'd'		→...

On cherche le courant politique dominant des élections (celui qui obtient le maximum de voix). Dans le cas de plus d'un, on dit qu'il n'y pas de courant dominant.

Exemple : D'après le tableau *T*, le courant politique gauche a dominé les élections par 11 voix.

8. Ecrire la fonction *dominant(T, nlr)* qui donne le courant politique dominant s'il existe sinon '_'.

Dans le cas d'absence de vainqueur majoritaire après 1^{er} et 2nd décomptage, une liste principale, ayant le maximum de voix, est chargée de former une coalition de listes réussie.

Une liste est réussie si elle est composée autour de la liste principale et totalise un score qui dépasse 50% des voix.

S'il y'a plus d'une liste de score maximum alors celle qui appartient au courant politique obtenant le plus de voix est désignée liste principale.

Exemple : dans T , la liste n°5 est la liste principale. Le score de son courant politique ('g') totalise 11 voix.

9. Ecrire la fonction $nvCourant1(T, nlr, j)$ qui retourne le total des voix du courant politique de la liste j .
10. Ecrire la fonction $listePrincipale(T, nlr)$ qui retourne le numéro de la liste principale.

Une coalition est représentée par un tableau C de taille nlr (nombre de liste restantes). $C[i]=1$ indique que la liste $T[i].nl$ participe à la coalition et $C[i]=0$ indique le contraire.

Exemple : La coalition formée des listes n°3 et n°5 suivante totalise 13 voix :

i	0	1	2	3
C	0	1	1	0

11. Écrire la fonction $score(T, C, nlr)$ qui retourne sc le score de la coalition représentée dans C .
12. Écrire la fonction $reussir(T, C, nlr, nvv)$ qui vérifie si la coalition représentée par C est réussie.

On calcule le total des voix obtenu par toutes les listes de coalition C de même courant politique que la liste j .

13. Ecrire la fonction $nvCourant2(T, C, nlr, j)$ qui retourne le score du courant politique de la liste j dans C .

Pour le maintien d'une coalition, celle-ci doit être stable. Une coalition est stable si elle est réussie et que toutes les listes participantes sont nécessaires pour dépasser les 50% des voix nécessaires.

Exemple : La coalition précédente est stable par contre la coalition suivante ne l'est pas car elle n'est réussie même sans la liste n°2 ou n°3 :

i	0	1	2	3
C	1	1	1	0

14. Écrire la fonction $stable(T, C, nlr, nvv)$ qui vérifie si la coalition dans C est stable ou pas.

IV- On veut aider la liste principale à former une coalition stable et le plus homogène possible, c'est-à-dire que la coalition est dominée par le courant politique de la liste principale (le maximum de voix de son courant politique).

Exemple : A partir de T , une coalition sera parfaite si la liste n°5 s'associe avec la liste n°2: $C = \{1, 0, 1, 0\}$.

On développe d'abord un générateur de coalitions sans condition préalable sur sa formation puis on sélectionne celle, supposée unique, répondant aux conditions et critères précédents (coalition parfaite).

Exemple : Les coalitions possibles, sans conditions préalables sur leur formation, relatives aux 4 listes précédentes sont au nombre de 15 :

$$\{0, 0, 0, 1\}, \{0, 0, 1, 0\}, \{0, 0, 1, 1\}, \{0, 1, 0, 0\}, \dots, \{1, 1, 1, 0\}, \{1, 1, 1, 1\}$$

On peut constater qu'une coalition peut être formée par la conversion en base 2 d'un entier x avec $1 \leq x \leq 15$

Exemple : Si $x = 3$ alors $C = \{0, 0, 1, 1\}$ et la coalition est formée des listes n°5 et n°6 et obtenant 10 voix.

15. Ecrire la fonction $g(nlr)$ qui donne le nombre de coalitions, sans condition, formées à partir de nlr listes.

16. Ecrire la fonction $generer(x, C, nlr)$ qui remplit C par la coalition correspondant à l'entier x ($1 \leq x \leq 15$).

17. Ecrire la fonction $coalParfaite(T, C, nlr, nvv)$ qui remplit C par la coalition parfaite.

V- Après formation d'une coalition, un conseil de Y membres est à constituer à partir des candidats de ses listes. Chaque liste électorale est composée de Y candidats présentés par le partie correspondant.

L'ordre des candidats dans la liste reflète leur importance et par conséquent augmente leur chance de devenir membre du conseil.

Le 1^{er} candidat d'une liste est tête de liste (le plus important), le suivant est le 2nd d'importance dans la liste, etc.

Exemple : Soit $Y=5$ et le tableau T précédent complété par les Y noms de candidats de chaque liste.

i	nl	sl	cl	L
0	2	4	'g'	→ 'xx01' → 'xx02' → 'xx03' → 'xx04' → 'xx05'
1	3	6	'c'	→ 'xx11' → 'xx12' → 'xx13' → 'xx14' → 'xx15'
2	5	7	'g'	→ 'xx21' → 'xx22' → 'xx23' → 'xx24' → 'xx25'
3	6	3	'd'	→ 'xx31' → 'xx32' → 'xx33' → 'xx34' → 'xx35'

Pour faciliter l'exemple, on représente les noms des candidats par la chaîne de caractères 'xx..'.
 Les candidats 'xx01', ..., 'xx31' ont plus de chance de siéger au conseil (devenir membre du conseil) qu'au reste des candidats.

Pour former le conseil, si une liste est vainqueur majoritaire alors ses Y candidats siégeront au conseil. Dans le cas de formation d'une coalition de liste, de chaque liste des candidats siégeront au conseil dans la limite des Y membres du conseil. Le nombre de candidats par liste participante à la coalition est proportionnel à son score sl .

Si la liste i de score j participant à la coalition C de score sc , alors $(j*Y)/sc$ de ses candidats siégeront au conseil. L'application de ce calcul à toutes les listes de C donnera un total de sièges $Y' < Y$ du fait de la division entière de $(j*Y)/sc$.

Les $Y-Y'$ sièges restants seront attribués un siège par liste aux listes qui ont le plus grand reste $(j*Y)\%sc$.

Les nombres de sièges attribués aux listes de la coalition sont représentés par un tableau d'entiers R de taille nlr .

Le conseil est représenté par une liste chaînée LC composée des noms de ses membres.

LC est déclarée globale. Sa déclaration est :

$listeC *LC = NULL ;$

Exemple : A partir du tableau T précédent et la coalition $C = \{1, 0, 1, 0\}$, la 1^{ère} attribution de sièges donnera 1 siège à la liste n°2 et 3 sièges à la liste n°5. Cependant, 1 siège restera ; celui-ci sera attribué à la liste n°2 qui a le plus grand reste. Ainsi, $R = \{2, 0, 3, 0\}$

Dans le conseil LC , on représente d'abord les têtes de listes participantes à la coalition puis le reste des candidats de chaque liste et tout cela dans l'ordre de leur apparition dans T (voir exemple)

Exemple : La liste du conseil LC est :

$LC \rightarrow \boxed{'xx01'} \rightarrow \boxed{'xx21'} \rightarrow \boxed{'xx02'} \rightarrow \boxed{'xx22'} \rightarrow \boxed{'xx23'}$

18. Ecrire la fonction $sieges(T, C, R, Y, nlr)$ qui remplit R par les nombres de sièges attribués aux listes de C .

19. Ecrire la fonction $conseil(T, R)$ qui forme la liste des membres du conseil LC .

Evaluation d'une expression algébrique

La notation habituelle des expressions algébriques, sous forme dite infixe, où les opérateurs +, *, -, / figurant entre leurs 2 opérands, souffre a priori d'une ambiguïté si l'on n'introduit pas de priorités entre les opérateurs. C'est ainsi que la notation $2 + 3 * 4$ peut aussi bien désigner $2 + (3*4) = 14$ que $(2+3) * 4 = 20$.

Des parenthèses ou des règles de priorité sont donc nécessaires pour lever cette ambiguïté. Nous allons étudier ici une autre notation, appelée notation algébrique post-fixée ou encore notation polonaise inversée qui ne souffre pas de ces inconvénients. Cette notation est utilisée par certains langages de programmation ou certaines calculatrices.

Exemple :

$$2\ 3 + \sin\ 4 * 5\ 6 + -$$

Son équivalent en notation infixée :

$$(\sin(2+3) * 4) - (5 + 6)$$

Les étapes de son évaluation :

+	opérateur binaire	$r = 2+3$
<i>sin</i>	un seul paramètre	$r = \sin(r)$
*	opérateur binaire	$r = r*4$
+	opérateur binaire	$s = 5+6$
-	opérateur binaire	$r = r - s$

On suppose dans ce problème une expression algébrique sous forme d'une chaîne de caractère C composée de : 4 opérateurs '+', '-', '*', '/', des opérands caractères appartenant à l'ensemble $\{'0', '1', \dots, '9'\}$ représentant des entiers entre 0 et 9 et des espaces de séparation.

La transformation d'une opérande caractère ' $0' \leq C[i] \leq '9'$ en entier équivalent se fait par la soustraction du caractère ' $0'$ de $C[i]$ (ex. $'1' - '0' = 1$)

Exemple :

$$5\ 2\ 3 * + 5\ 6 + -$$

Les étapes de son évaluation :

$$\begin{aligned} r &= 2*3 \\ r &= r + 5 \\ s &= 5+6 \\ r &= r - s \end{aligned}$$

Des étapes de l'évaluation, on remarque que les opérands de C sont mémorisés puis évalués par les opérateurs de C suivant le principe LIFO (Last In First Out) implémenté par une pile d'entiers L.

1. Ecrire les fonctions *empiler(L, val)*, *depiler(L)* et *sommet(L)* qui respectivement ajoute à la pile une opérande entier, supprime une opérande de la pile et donne l'opérande se trouvant à son sommet.

N.B. vous pouvez définir les fonctions *empiler(...)* et *depiler(...)* autrement.

On supposant que l'expression C est rédigée correctement selon la notation algébrique post-fixée.

2. Ecrire la fonction *int eval(C)* qui donne la valeur de l'expression algébrique C.

Matrice creuse

- ✓ On attachera une importance à la concision, à la clarté, et à la précision de la rédaction en langage C.
- ✓ La fonction prédéfinie autorisée est *printf*.

Une matrice creuse est une matrice contenant plus de 50% de valeurs nulles. La représentation d'une matrice par un tableau à 2 dimensions, conduit à une occupation inutile de l'espace mémoire par des 0.

Dans ce problème, on travaille sur d'autres moyens de représentation des matrices creuses.

Soit une matrice d'entiers M de taille $N \times N$ déclarée et initialisée dans la fonction *main()*. N est une constante globale.

Exemple : matrice M de taille 5×5 :

5	4	0	0	1
0	5	0	0	0
0	1	0	7	0
0	0	0	0	0
0	0	3	0	0

Proposition 1

La proposition 1 consiste à représenter la matrice M par deux tableaux Tv et Tn . Le tableau Tv contient les couples : valeur non nulle de M et son numéro de colonne. Le tableau Tn contient le nombre de valeurs non nulles de M par ligne.

Exemple : D'après l'exemple précédent Tv est :

5	4	1	5	1	7	3
0	1	4	1	1	3	2

et Tn est :

3	1	2	0	1
---	---	---	---	---

1. Définir le type du tableau d'enregistrements Tv .
2. Sans programmer, que pouvez-vous dire sur la dimension du tableau Tv et du tableau Tn ?

On cherche à transformer M en Tv et Tn .

3. Ecrire une fonction *nombreV(M)* qui donne le nombre de valeurs non nulles de M .
4. Ecrire la fonction *remplirTv(M)* qui retourne le tableau Tv obtenu à partir de M .
5. Ecrire la fonction *remplirTn(M, Tn)* qui remplit le tableau Tn à partir de M .

On veut exploiter la matrice M à travers sa représentation équivalente Tv et Tn .

Répondre aux questions suivantes à partir de Tv et Tn au lieu de M .

6. Ecrire la fonction *sommeCI(Tv, Tn, j)* qui retourne la somme de la colonne j .
7. Ecrire la fonction *sommeLI(Tv, Tn, i)* qui retourne la somme de la ligne i .
8. Ecrire la fonction *valeurI(Tv, Tn, i, j)* qui donne la valeur de $M[i][j]$.
9. Ecrire la fonction *traceI(Tv, Tn)* qui donne la trace de la matrice M .

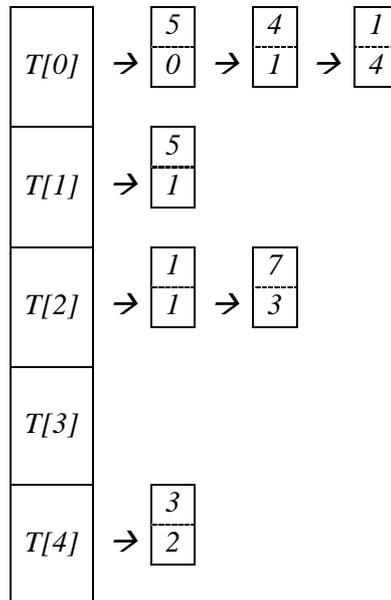
On veut afficher les valeurs de la ligne i de M . Si $i = 2$ alors on affiche $0\ 1\ 0\ 7\ 0$

10. Ecrire la fonction *afficheLI(Tv, Tn, i)* qui affiche les valeurs de la ligne i de M .

Proposition 2

Consiste à représenter la matrice M par un tableau de listes chaînées T de taille N où chaque liste chaînée $T[i]$ est composée des valeurs non nulles de la ligne i de M et les numéros de colonnes correspondantes. T est déclaré dans la fonction $main()$ et initialisé à la valeur $NULL$ ($T[i] = NULL$).

Exemple : D'après l'exemple précédent T est :



11. Définir le type du tableau de listes T et donner la déclaration du tableau T .

On veut transformer la matrice M en tableau de listes T .

12. Ecrire la fonction $remplirT(M, T)$ qui remplit le tableau T à partir de M .

On cherche à exploiter la matrice M à travers sa représentation équivalente T .

Répondre aux questions suivantes à partir de T au lieu de M .

13. Ecrire la fonction $sommeL2(T, i)$ qui retourne la somme de la ligne i .

14. Ecrire la fonction $sommeC2(T, j)$ qui retourne la somme de la colonne j .

15. Ecrire la fonction $valeur2(T, i, j)$ qui retourne la valeur de $M[i][j]$.

16. Ecrire la fonction $trace2(T)$ qui retourne la trace de la matrice M .

On veut afficher les valeurs de la ligne i de M .

Exemple : Si $i = 2$ alors on affiche $0\ 1\ 0\ 7\ 0$

17. Ecrire la fonction $afficheL2(T, i)$ qui affiche les valeurs de la ligne i de M .

On développe les fonctions permettant la transformation de T en Tv et Tn et inversement sans passer par M .

18. Ecrire la fonction $tabListe(Tv, Tn, T)$ qui remplit T à partir de Tv et Tn .

19. Ecrire la fonction $listeTab(T, Tv, Tn)$ qui remplit Tv et Tn à partir de T .

Le jeu du SuDoKu

- ✓ On attachera une importance à la concision, à la clarté, et à la précision de la rédaction en langage C.
- ✓ Aucune fonction prédéfinie n'est autorisée.
- ✓ a et b deux entiers positifs, a/b donne le quotient de la division et $a\%b$ donne son reste.

On dispose d'une grille SuDoKu sous forme d'une matrice d'entiers T de taille $N \times N$ partiellement remplie par des valeurs entre 1 et 9. T est subdivisée en 9 carrés de taille 3×3 .

Un carré est l'intersection entre 3 lignes et 3 colonnes de T (voir exemple).

N est une constante globale égale 9.

I- Description du jeu

Le jeu consiste à remplir les cases vides de T avec un entier entre 1 et 9 de sorte que l'entier n'apparaît que :

- une seule fois dans une ligne de T
- une seule fois dans une colonne de T .
- une seule fois dans un carré de T .

Exemple de grille :

i/j	0	1	2	3	4	5	6	7	8
0	5	3			7				
1	6			1	9	5			
2		9	8					6	
3	8				6				3
4	4			8		3			1
5	7				2				6
6		6					2	8	
7				4	1	9			5
8					8			7	9

Exemple : Le carré obtenu par l'intersection des 3 dernières lignes et les 3 dernières colonnes est :

i/j	6	7	8
6	2	8	
7			5
8		7	9

1. Compléter le remplissage des cases vides ce carré. Les entiers possibles sont : 1, 3, 4 et 6.

Dans ce problème nous développons des fonctions, qui permettent de compléter le remplissage de T .

II- Manipulation de T

T est une matrice déclarée et initialisée dans la fonction $main()$.

Les cases vides de T sont initialisées à la valeur 0.

On suppose que l'initialisation mène à une solution de la grille SuDuKu T .

Exemple :

i/j	0	1	2	3	4	5	6	7	8
0	5	3	0	0	7	0	0	0	0
1	6	0	0	1	9	5	0	0	0
2	0	9	8	0	0	0	0	6	0
3	8	0	0	0	6	0	0	0	3
4	4	0	0	8	0	3	0	0	1
5	7	0	0	0	2	0	0	0	6
6	0	6	0	0	0	0	2	8	0
7	0	0	0	4	1	9	0	0	5
8	0	0	0	0	8	0	0	7	9

2. Ecrire la fonction $nbVide(T)$ qui retourne le nombre de cases vides dans T .

Pour identifier les lignes et les colonnes de T , on utilise respectivement les indices i et j .

Une case de T est identifiée soit par ses indices i et j ($T[i][j]$) ou par son ordre k dans T ($0 \leq k < 9 \times 9$) où k est ordonné de gauche à droite et de haut vers le bas.

La relation entre k et i et j est $k = ixN + j$.

Ainsi, $i = k/N$ et $j = k \% N$.

Exemple : Les cases 0, 8, 9, 80 de T ont respectivement les indices (i, j) suivants : (0, 0), (0, 8), (1, 0), (8, 8).

3. Ecrire la fonction $f(i, j)$ qui retourne k correspondant aux indices i et j d'une case de T .

Un carré de T est identifié par la position k de la case de T qui se trouve à son centre.

Exemple : Les carrés 1, 2, ..., 9 de T sont identifiés par les valeurs de k suivantes : 10, 13, 16, 37, 40, 43, 64, 67, 70 correspondant aux indices i et j suivants : (1, 1), (1, 4), (1, 7), (4, 1), (4, 4), (4, 7), (7, 1), (7, 4), (7, 7).

4. Ecrire la fonction $carre(T, i, j)$ qui retourne k le numéro du carré où se trouve la case d'indices i et j .

III-Remplissage de T

Un entier x entre 1 à 9 est possible pour $T[i][j]=0$, si x n'est pas déjà dans la ligne i , ou dans la colonne j , ou dans le carré k contenant $T[i][j]$.

5. Ecrire la fonction $verifL(T, i, x)$ qui vérifie si x se trouve dans la ligne i ou pas.
6. Ecrire la fonction $verifC(T, j, x)$ qui vérifie si x se trouve dans la colonne j ou pas.
7. Ecrire la fonction $verifR(T, i, j, x)$ qui vérifie si x se trouve dans le carré contenant la case $T[i][j]$.
8. Ecrire la fonction $estPossible(T, i, j, x)$ qui vérifie si x est possible dans $T[i][j]$.

Pour mémoriser les entiers x possibles dans chaque $T[i][j]$, on utilise une matrice d'entiers P de taille $M \times N$ remplie par des 0 et 1.

M est une constante globale égale à $81 = 9 \times 9$.

L'indice ligne de P est k (position de $T[i][j]$ dans T) et son indice colonne est $l = x - 1$.

Exemple : $P[k][x-1] = 1$ signifie que l'entier x est possible dans $T[i][j]$ et $P[k][x-1] = 0$ signifie le contraire.

P est déclarée dans la fonction $main()$.

Exemple : La Grille T (à gauche) et sa matrice de possibilité P (à droite). En raison de la taille de P qui est grande, on ne représente que ses lignes qui correspondent au carré 9 de T .

i/j	0	1	2	3	4	5	6	7	8
0	5	3			7				
1	6			1	9	5			
2		9	8					6	
3	8				6				3
4	4			8		3			1
5	7				2				6
6		6					2	8	
7				4	1	9			5
8					8			7	9

k/l	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
...
60	0	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	0	0	0
62	0	0	0	1	0	0	0	0	0
...
78	1	0	1	1	0	1	0	0	0
79	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0

$T[7][7]$ correspondant à la ligne 70 de P ne peut recevoir que l'unique valeur 3.

$T[8][6]$ correspondant à la ligne 78 de P peut recevoir les valeurs 1, 3, 4 et 6.

$T[8][8]$ correspondant à la ligne 80 de P ne peut rien recevoir. Elle est déjà rempli.

9. Ecrire la fonction $init(P)$ qui initialise la matrice P à la valeur 0.
10. Ecrire la fonction $remplirP(T, P)$ qui remplit P par les entiers possibles dans les cases vides de T .

Afin de compléter le remplissage de T , on suit la stratégie suivante :

- Etape1 : On remplit la matrice des possibilités P à partir de la grille T .
- Etape2 : On commence par chercher dans P , les valeurs possibles uniques puis on les affecte aux cases vides de T correspondantes.
- On recommence le remplissage de la nouvelle grille T obtenue après les étapes 1 et 2 et cela jusqu'au remplissage complet de T .

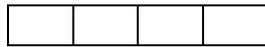
11. Ecrire la fonction $val(P, T, i, j)$ qui retourne l'entier possible de $T[i][j]$ s'il est unique sinon 0.
12. Ecrire la fonction $remplirT(T, P)$ qui complète le remplissage de T .
13. Donner une version récursive de la fonction $remplirT$.

On veut optimiser le temps de réponse de cette stratégie. Pour cela, et au lieu de répéter le remplissage de P à chaque fois qu'on affecte un entier à une case vide de T , on ne modifiera que les lignes de P concernées par le remplissage de la case vide de T .

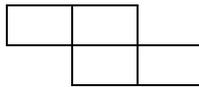
14. Sans programmer, proposer une stratégie alternative qui se base sur les remarques précédentes.
15. Ecrire la fonction $remplirT2(T, P)$ qui complète le remplissage de T selon cette stratégie.

Le jeu de Tetris

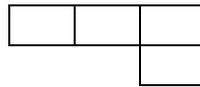
Tetris est un jeu vidéo célèbre qui utilise les 5 pièces suivantes:



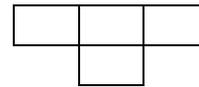
Pièce 1



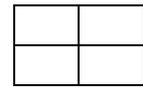
Pièce 2



Pièce 3



Pièce 4



Pièce 5

Dans ce problème, nous avons une grille de $N \times N$ entiers ($4 \leq N \leq 100$) (exemple Figure 1 pour $N=4$). Nous voulons placer une seule pièce de Tetris sur la grille telle que la somme des nombres au-dessous de la pièce est maximale.

Noter que toutes les pièces de Tetris, sauf la dernière, peuvent être tournées de 90° . Quelques pièces peuvent avoir jusqu'à quatre orientations différentes.

70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

Figure 1

70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

Figure 2

70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

Figure 3

70	2	1	7
7	1	30	6
4	30	30	5
3	1	30	2

Figure 4

N'importe quelle orientation est acceptable tant que la pièce est entièrement à l'intérieur de la grille. Par exemple, la pièce 1 peut être placée sur la 1^{ère} ligne de la grille (Fig2), avec une somme de 80.

Elle peut également être placée, par exemple, sur la 3^{ème} colonne, rapportant une somme de 91 (Fig3).

En fait, dans une grille 4×4 , nous pouvons avoir 77 manières de placer les pièces de Tetris.

Dans l'exemple, la plus grande somme est réalisée grâce à la pièce 4 avec une somme de 120 (Fig4).

On considère une grille T de taille $N \times N$. N est une constante globale.

1. Ecrire les fonctions $f1$, $f19$, $f2$, $f29$, $f3$, $f39$, $f318$, $f327$, $f4$, $f49$, $f418$, $f427$ et $f5$ qui retournent respectivement les sommes obtenues par l'application des pièces : pièce 1 et sa rotation de 90° ; pièce 2 et sa rotation de 90° ; pièce 3 et ses rotations de 90° , 180° et 270° , pièce 4 et ses rotations de 90° , 180° et 270° et enfin la pièce 5 sur la grille.
2. Écrire une fonction qui retourne la plus grande somme de T .

Tri de chaînes de caractères

On considère N chaînes de caractères de taille maximale M stockées dans une matrice T .

N et M sont des constantes globales.

Ecrire une fonction qui prend en paramètre T et qui permet de trier et d'afficher les chaînes de la manière suivante :

On trie les caractères de chaque chaîne puis on trie les chaînes entre elles.

Exemple 1 :

<u>Avant tri :</u>	<u>Après tri :</u>
"znb"	"ar"
"cd"	"bnz"
"z"	"cd"
"ra"	"z"

Exemple 2 :

<u>Avant tri :</u>	<u>Après tri :</u>
"elk"	"ay"
"rmoze"	"efz"
"ezf"	"ekl"
"ya"	"emorz"
"umh"	"hmu"
"ijkl"	"ijkl"

Voyage à prix bas

Les prix de voyages par avion sont fous ! Le prix d'un billet est déterminé par plusieurs facteurs, et n'est pas d'habitude directement lié à la distance du voyage. Les agences de voyages essaient d'être innovatrices, utilisant parfois des billets pouvant servir à différentes destinations afin de proposer des voyages à prix bas.

Pourtant, les compagnies aériennes sont conscientes de ce comportement, et d'habitude exigent qu'un billet couvrant un voyage le soit dans l'ordre indiqué des villes.

Par exemple, si vous avez un billet pour voyager d'une ville 1 à une ville 2 puis à la ville 3, vous n'êtes pas autorisé à utiliser seulement le tronçon de billet de la ville 2 à la ville 3. Vous devrez toujours commencer de la première ville mentionnée sur le billet. En plus, vous n'êtes pas autorisé à voyager de la ville 1 à la ville 2, voler quelques parts ailleurs, revenir puis continuer votre vol de la ville 2 à la ville 3.

Prenons un exemple, supposons qu'une compagnie aérienne propose trois types de billets :

Billet #1 : Ville 1 à Ville 3 à Ville 4	225.00 Dh
Billet #2 : Ville 1 à Ville 2	200.00 Dh
Billet #3 : Ville 2 à Ville 3	50.00 Dh

Supposons que vous voulez voyager de la ville 1 à la ville 3, il y a deux façons d'y arriver en utilisant seulement les choix possibles de billets.

1^{er} choix : Billet #1 au prix de 225.00 Dh en utilisant que le premier tronçon du billet.

2^{ème} choix : Billet #2 au prix de 200.00 Dh et Billet #3 au prix de 50.00 Dh

Le 1^{er} choix est le moins cher.

1. Ecrire une fonction qui permet de remplir une matrice M , à partir des données du fichier d'entrée.
2. Donnons une suite d'offres de billets, un ou plusieurs itinéraires de voyages, vous devez déterminer comment avoir des billets à prix bas.

Fichier d'entrées :

Les données d'entrées consistent dans de multiples tests, chacun décrit une suite d'offres de billets et une suite d'itinéraires de voyages.

Chaque test commence par une ligne contenant nombre d'offres de billets (NB), suivi par NB descriptions d'offres, une par ligne.

Chaque description est composée d'un nombre entier positif spécifiant le prix du billet, le nombre de villes traversées, et les villes. Chaque ville est identifiée par un nombre entier unique.

La ligne qui suit les itinéraires contient le nombre de voyages à vouloir à prix bas (NV), suivie par NV descriptions de voyage. Chaque ligne comprend le nombre de villes dans le voyage y compris la ville de départ, suivi par les numéros des villes, donnés dans l'ordre de leur visite.

Dans le cas du test, les offres de billets et de voyage ne dépasseront pas 20. Chaque voyage ou billet aura une liste de villes de 1 à 9.

Le dernier cas est suivi par une ligne contenant 0.

Fichier de sortie

Pour chaque voyage, les deux lignes de sortie contiennent le numéro de test, le numéro de voyage, le prix minimum, et les numéros de billets utilisés dans le voyage dans l'ordre de leur utilisation.

Suivre le format de sortie indiqué ci-contre.

Exemple de données d'entrées	Exemple de données de sorties correspondant
3 225 3 1 3 4 200 2 1 2 50 2 2 3 1 2 1 3 3 100 2 2 4 100 3 1 4 3 200 3 1 2 3 2 3 1 4 3 3 1 2 4 0	Test 1, Voyage 1: Prix = 225.00 Dh Billets utilisés : 1 Test 2, Voyage 1: Prix = 100.00 Dh Billets utilisés : 2 Test 2, Voyage 2: Prix = 300.00 Dh Billets utilisés : 3 1

Filtre non causal

I- Lissage d'un signal :

L'objectif de cet algorithme est de développer un filtre non causal, c'est-à-dire une fonction qui lisse un signal (considéré comme un tableau de valeurs) en utilisant une fenêtre glissante d'une taille n pour moyenner les valeurs du signal. Une valeur filtrée p est la moyenne des valeurs non encore filtrées se trouvant dans une fenêtre centrée en p .

On considère un tableau non filtré Tnf et un tableau filtré Tf de taille N (déclaré constante globale)

Exemple :

Un signal de valeurs :

2	1	4	5	3	6	3	7
---	---	---	---	---	---	---	---

Devient après filtrage:

1.5	2.3	3.3	4	4.7	4	5.3	5
-----	-----	-----	---	-----	---	-----	---

1. Ecrire la fonction $filtreSig(Tnf, Tf)$ qui filtre le signal Tnf dans Tf .

II- Lissage d'une image

A l'image du filtre non causal, cette fonction calculera la couleur d'un point p de la nouvelle image en faisant la moyenne des couleurs des points de l'image source se trouvant dans une fenêtre carrée centrée en p .

La figure ci-dessous propose un exemple de calcul d'une image de largeur 4 et de hauteur 5 avec un filtre de taille 3.

147	14	97	89
234	95	1	187
153	17	156	137
230	73	135	79
37	232	32	48

Image source

122.5	98	80.5	93.5
110	101.6	88.1	111.2
133.7	121.6	97.8	115.8
123.7	118.3	101	97.8
143	123.2	99.8	73.5

Image filtrée

Soit une matrice non filtré Wnf et une matrice filtré Wf de taille $N \times M$ (déclarés constantes globales)

2. Ecrire la fonction $filtreImage(Wnf, Wf)$ qui filtre l'image Wnf dans Wf .

III- Compression d'un signal

On peut considérer qu'un signal est une suite de valeurs représentant les amplitudes du signal. Une méthode de compression simple est de remplacer cette suite de valeurs par une suite de couples de valeur (occurrence, amplitude).

Par exemple la suite 10, 10, 10, 4, 4, 5, 5, 5, 5, 8 peut donner la suite 3, 10, 2, 4, 4, 5, 1, 8 (on a ici économisé 2 valeurs). La taille du tableau Tc , représentant le signal, est très grande. La fin du signal est identifiée par la valeur -1.

3. Ecrire la fonction $comprime(Tc)$ qui permet de compresser le signal Tc dans le même tableau.

IV- Qualité du signal et compression.

La fonction précédente permet de récupérer le signal identique à l'original après décompression.

4. Récrire la fonction $comprime(Tc)$ qui garde une qualité de 80% du signal d'origine après décompression (c'est-à-dire que si 2 points successifs qui diffèrent de 20% alors ils sont supposés égaux et la 1^{ère} valeur est compressée).

Réseau d'ordinateurs

Partie 1 : Modélisation du réseau

I- On considère un réseau informatique composé d'un ensemble d'ordinateurs reliés entre eux. Chaque ordinateur est caractérisé par un nom et possède N ports de Liaison qui y sont rattachés (ex. ports USB) et qui permettent de connecter l'ordinateur à d'autres ordinateurs.

Ainsi, une Liaison relie deux ordinateurs. Elle est caractérisée par un débit (entier) et ses deux ordinateurs d'extrémités rattachés.

Nous allons définir un type Ordinateur et un type Liaison.

D'un point de vue de la représentation mémoire, il est impossible que l'Ordinateur contienne les Liaisons, et qu'une Liaison contienne les Ordinateurs.

Un Ordinateur contient donc un ensemble de pointeurs sur des variables de type Liaison et une Liaison aura des pointeurs sur ces deux extrémités de type Ordinateurs.

Chaque ordinateur ne peut être connecté qu'au maximum N autres ordinateurs. Un port non connecté est de valeur *NULL*.

1. Proposer les types de données utilisées dans ce problème.
2. Proposer un exemple d'implémentation mémoire du réseau.

On supposant un tableau d'ordinateurs T de taille M (M ordinateurs).

3. Écrire une fonction qui retourne le nombre de connexion effective dans le réseau.

Partie 2 : Détection d'un Fanion Début et Fin

Une liaison permet une transmission de données (des bits 0 et 1) entre ses deux ordinateurs d'extrémités. La liaison est de type série permettant de transmettre bit par bit les données.

La séquence de bits qui transite, sur la liaison, d'un ordinateur X à un ordinateur Y est représentée par un tableau d'entiers T de très grande taille de valeurs 0 et 1.

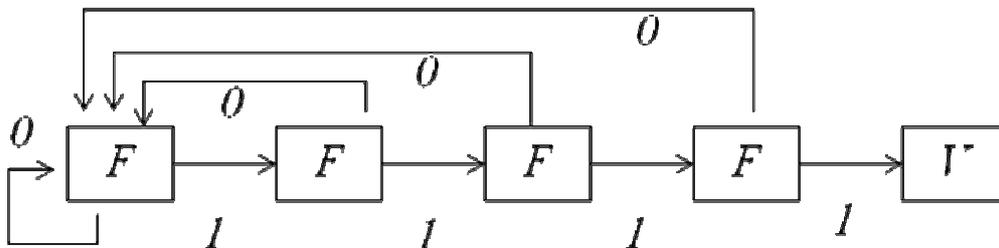
T contient, au début, une séquence continue de bits inutiles (absence de données à transmettre de X vers Y), suivie d'une séquence de bits pour détecter le début de la transmission (appelée Fanion Début), suivie d'une séquence de bits utiles (début de transmission), suivie d'une séquence de bits pour détecter la fin de la transmission de donnée (Fanion Fin), suivie de séquence de bits inutiles, etc.

Le Fanion début est identique au Fanion fin.

La suite de bits permettant de détecter le début et la fin de la transmission est $\{1, 1, 1, 1\}$

Remarque : pour éviter de confondre une suite de bits utile identique à celle du Fanion, l'émetteur code ses données à transmettre de sorte de ne pas avoir une telle suite au sein de ses données.

Pour détecter le Fanion, on propose l'automate A suivant : (partie concernant le Fanion début)



F : indique Faux (Fanion début non détecté)

V : indique Vrai (détection du Fanion début).

0 et 1 sont les bits du tableau T transits de l'ordinateur X vers Y.

N.B. L'ordre de transmission de bits est l'ordre croissant des indices de T .

1. Proposer le type de A .
2. Ecrire une fonction qui permet de l'implémenter en mémoire.
3. Ecrire une fonction qui prend en paramètre le tableau T et l'automate A et retourne 1 s'il détecte le Fanion sinon 0.
4. Sans programme, proposer le schéma complet de l'automate A qui permet de détecter le début et la fin de la transmission.

Bonne chance