UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

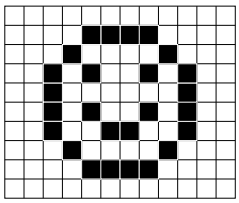
Licence 1 INFO
INFO0101

TP n°10

Les tableaux à deux dimensions

1 Image en noir et blanc

Nous souhaitons représenter une image en noir et blanc à l'aide d'un tableau à 2 dimensions de booléens (nous considérons ici que l'image est un tableau de lignes). Chaque case correspond à un pixel qui peut être allumé (valeur `vrai`) ou éteint (valeur `faux`). La figure suivante montre un exemple d'image *(a)*, ainsi qu'un exemple d'affichage dans l'invite de commandes *(b)* :



(a)

```
____________
____XXXX____
___X____X___
__X_X__X_X__
__X_____X__
__X_X__X_X__
__X__XX__X__
___X____X___
____XXXX____
____________
```

(b)

1. Écrivez une fonction/procédure `effacerImage` qui prend en paramètre une image et qui efface tous les pixels.
2. Écrivez une fonction/procédure `creerImage` qui prend en paramètre une largeur et une hauteur et qui crée une image vide (sans pixel allumé).
3. Écrivez une fonction/procédure `afficherImage` qui prend en paramètre une image et qui l'affiche à l'écran : un pixel éteint est représenté par un espace et un pixel allumé par un "X".

Vous pouvez téléchargez sur *Moodle* le fichier `image.txt` contenant l'image précédente.

4. Écrivez une fonction/procédure `setPixel` qui prend en paramètres les coordonnées d'un pixel et une image. Si les coordonnées sont correctes, le pixel est allumé.
5. Écrivez une fonction/procédure `ligne` qui prend en paramètres l'indice d'une ligne et une image. Elle allume tous les pixels de la ligne concernée. Vous devez traiter les cas où l'indice passé en paramètre est incorrect.
6. Nous souhaitons écrire une fonction `cadre` qui retourne les coordonnées du rectangle encadrant les pixels allumés de l'image. Par exemple, sur l'image précédente, le rectangle possède les coordonnées (1,2)-(8,9) (ce sont les coordonnées du point en haut à gauche et du point en bas à droite). En pratique, les coordonnées d'un rectangle sont représentées par un tableau à 2 dimensions, chaque ligne correspondant aux coordonnées d'un point.

   — Écrivez la fonction `premiereLigne` qui retourne l'indice de la première ligne contenant un pixel allumé (avec l'exemple précédent, elle retourne 1). Si l'image est vide, la fonction retourne 0.

- Écrivez la fonction `premiereColonne` qui retourne l'indice de la première colonne contenant un pixel allumé.
- Écrivez la fonction `derniereLigne` qui retourne l'indice de la dernière ligne contenant un pixel allumé (avec l'exemple précédent, elle retourne 8). Si l'image est vide, la fonction retourne l'indice de la dernière ligne.
- Écrivez la fonction `derniereColonne` qui retourne l'indice de la dernière colonne contenant un pixel allumé (avec l'exemple précédent, elle retourne 9). Si l'image est vide, la fonction retourne l'indice de la dernière colonne.
- Écrivez la fonction `cadre`.

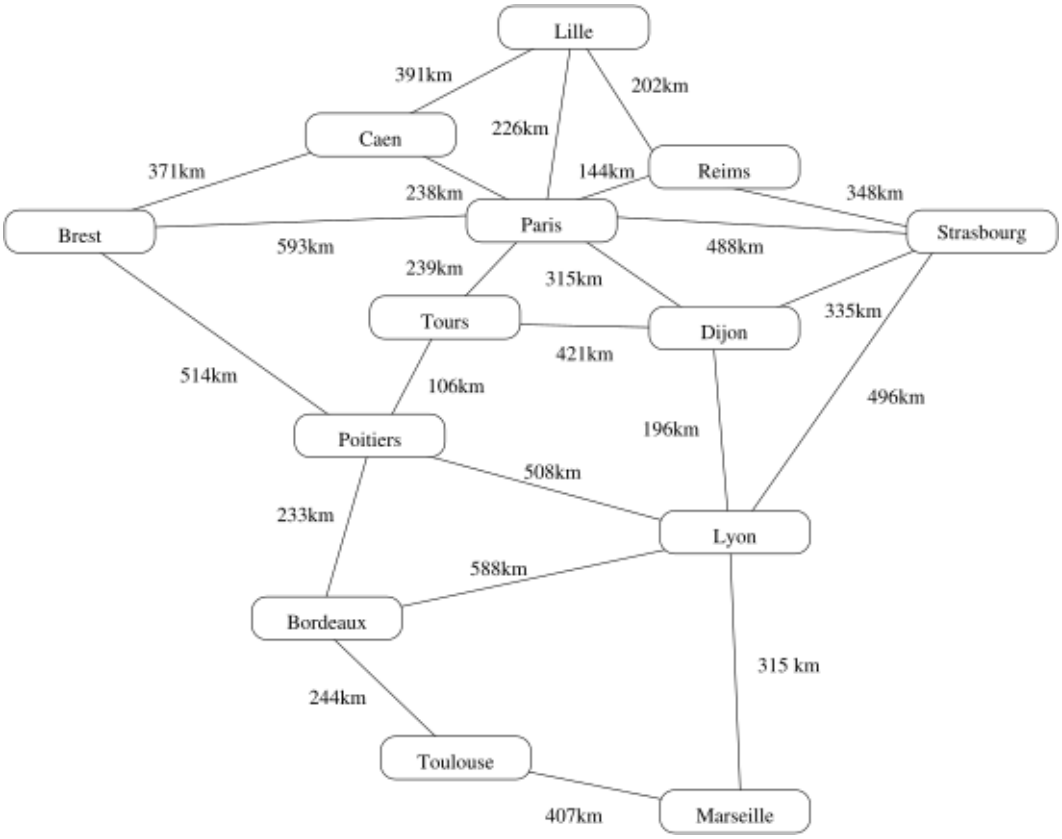
7. Écrire une fonction/procédure `inverser` qui prend en paramètres une image et les coordonnées d'un rectangle et qui inverse, dans l'image, tous les pixels qui font partie du rectangle. Vous gèrerez les cas d'erreur.
8. Écrivez un `main` qui permet de tester **toutes** les fonctions/procédures réalisées précédemment.

2 Google Map (en mieux)

On souhaite écrire une application pour calculer la distance de trajets entre des villes. Nous partons sur un ensemble de villes connues et des distances entre ces différentes villes. L'utilisateur peut réaliser un trajet qui correspond à une ville de départ et une ville d'arrivée, ainsi qu'une ou plusieurs villes étapes

Pour représenter les villes, nous utilisons un tableau de chaînes de caractères. Chaque case correspond à une ville et l'indice permettra de représenter par la suite cette ville. Par exemple, si la ville de Reims est située dans la case d'indice 9, donc 9 représente la ville de Reims.

Pour représenter les distances entre les villes, nous utilisons une matrice de n cases sur n cases, avec n le nombre de villes. La valeur de la case (9, 7) correspond à la distance entre la ville 9 (Reims) et la ville 7 (Paris). Si la valeur vaut 0, c'est qu'il n'y a pas de route directe entre les villes, sinon la valeur correspond à un nombre de kilomètres. Pour exemple, voici la carte que nous souhaitons représenter :



Le tableau des villes est donc le suivant :

0	Bordeaux	3	Dijon	6	Marseille	9	Reims	12	Tours
1	Brest	4	Lille	7	Paris	10	Strasbourg		
2	Caen	5	Dijon	8	Poitiers	11	Toulouse		

La matrice des distances est donc :

0	0	0	0	0	588	0	0	233	0	0	244	0
0	0	371	0	0	0	0	593	514	0	0	0	0
0	371	0	0	391	0	0	238	0	0	0	0	0
0	0	0	0	0	196	0	315	0	0	335	0	421
0	0	391	0	0	0	0	226	0	202	0	0	0
588	0	0	196	0	0	315	0	508	0	496	0	0
0	0	0	0	0	315	0	0	0	0	0	407	0
0	593	238	315	226	0	0	0	0	144	488	0	239
233	514	0	0	0	508	0	0	0	0	0	0	106
0	0	0	0	202	0	0	144	0	0	348	0	0
0	0	0	335	0	496	0	488	0	348	0	0	0
244	0	0	0	0	0	407	0	0	0	0	0	0
0	0	0	421	0	0	0	239	106	0	0	0	0

1. Écrire `saisirVilles` qui demande à l'utilisateur un nombre de villes et qui lui demande de saisir les noms de chacune d'entre elles.
2. Écrire `afficherVilles` qui affiche un tableau de villes (vous n'oublierez pas d'afficher l'indice correspondant à chacune d'elles).
3. Écrire `saisirDistances` qui demande à l'utilisateur de saisir les distances entre les villes.
4. Écrire `afficherDistances` qui affiche la matrice des distances.

5. Écrire `saisirParcours` qui demande à l'utilisateur de saisir un parcours. Un parcours est un tableau d'entiers de longueur quelconque, chaque case correspondant à un numéro de ville. Bien entendu, il faut vérifier que le parcours est valide.

Pour éviter de saisir les villes à chaque fois, vous pouvez utiliser un tableau contenant les villes et les distances.

Vous pouvez téléchargez les extraits de code suivants sur *Moodle*.

Par exemple avec trois villes :

```
String villes[] = { "Reims", "Paris", "Strasbourg"};
int distances[][] = { {0, 144, 348}, {144, 0, 488}, {348, 488, 0}};
```

Si vous préférez l'exemple complet avec les 13 villes :

```
String villes2[] = {
    "Bordeaux", "Brest", "Caen", "Dijon", "Lille", "Lyon",
    "Marseille", "Paris", "Poitiers", "Reims", "Strasbourg",
    "Toulouse", "Tours"
  };
int distances2[][] = {
    {0, 0, 0, 0, 0, 588, 0, 0, 233, 0, 0, 244, 0},
    {0, 0, 371, 0, 0, 0, 0, 593, 514, 0, 0, 0, 0},
    {0, 371, 0, 0, 391, 0, 0, 238, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 196, 0, 315, 0, 0, 335, 0, 421},
    {0, 0, 391, 0, 0, 0, 0, 226, 0, 202, 0, 0, 0},
    {588, 0, 0, 196, 0, 0, 315, 0, 508, 0, 496, 0, 0},
    {0, 0, 0, 0, 0, 315, 0, 0, 0, 0, 0, 407, 0},
    {0, 593, 238, 315, 226, 0, 0, 0, 0, 144, 488, 0, 239},
    {233, 514, 0, 0, 0, 508, 0, 0, 0, 0, 0, 0, 106},
    {0, 0, 0, 0, 202, 0, 0, 144, 0, 0, 348, 0, 0},
    {0, 0, 0, 335, 0, 496, 0, 488, 0, 348, 0, 0, 0},
    {244, 0, 0, 0, 0, 0, 407, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 421, 0, 0, 0, 239, 106, 0, 0, 0, 0}
  };
```

6. Écrire `afficherParcours` qui permet d'afficher un parcours.
7. Écrire `distanceParcours` qui permet de calculer la distance d'un parcours.
8. Écrivez un `main` complet pour tester votre application.