TP n°18 – Programmation dynamique

Des fichiers avec des fonctions utilitaires d'affichage sont disponibles (pour les exercices en C et en Ocaml)

Exercice 1 : Plus grande somme consécutive (en C)

On se donne un tableau `val` d'entiers relatifs ; on veut déterminer la plus grande somme possible d'entiers consécutifs de `val`. On note n la taille de `val`.

Par exemple dans `[4;-123;5;6;-1;101]`, la plus grande somme d'entiers consécutifs est $111 = 5 + 6 - 1 + 101$. On veut utiliser une approche par programmation dynamique.

- **Q1.** Sans regarder la ligne d'en dessous, réfléchir à la sous-structure optimale. (c'est à dire quelle variable d'entrée on va pouvoir limiter pour simplifier le problème)

On introduit la fonction $S_{max}(i) = \max_{0 \leq k \leq i} \Sigma_{j=k}^{i} \texttt{val}[j]$.

- **Q2.** Déterminer une formule de récurrence pour $S_{max}(i)$. Indice : cela fait intervenir $S_{max}(i-1)$ et `val[i]`.
- **Q3.** Comment obtenir la plus grande somme d'entiers consécutive dans `val` à partir des valeurs de S_{max} ?
- **Q4.** Programmer l'approche ascendante. Ici il n'y a qu'un paramètre dans notre fonction récursive, donc on pourra utiliser un tableau.

Avec la méthode précédente, on a obtenu la valeur de la plus grande somme, mais pas entre quels indices elle est calculée. On peut savoir entre quels indices calculer cette somme en regardant S_{max}.

- **Q5.** Écrire une fonction qui prend en entrée un tableau correspondant aux valeurs de S_{max} et renvoie les indices entre lesquels la plus grande somme est calculée.
  Si on reprend l'exemple introductif de l'exercice, le retour attendu est `[2,5]`. On met les indices dans un tableau de taille 2 pour pouvoir les renvoyer.

Exercice 2 : Project Euler, problem 67 (en Ocaml)

On considère un triangle de nombres de hauteur n. On cherche la somme maximale des valeurs sur un chemin reliant le sommet à la base du triangle. On pourra considérer que les valeurs de ce triangle sont stockées dans un tableau bi-dimensionnel $n \times n$ (autrement dit, `t.(i).(j)` contient la $(j+1)e$ valeur de la $(i+1)e$ ligne).

Par exemple, en partant du sommet du triangle ci-dessous et en se déplaçant vers les nombres adjacents de la ligne inférieure, le total maximum que l'on peut obtenir pour relier le sommet à la base est égal à 23 : 3 + 7 + 4 + 9 = 23

```
   3
  7 4
 2 4 6
8 5 9 3
```

On stockera la pyramide dans une matrice de la manière suivante :

```
[|[|3; 0; 0; 0|];
 [|7; 4; 0; 0|];
 [|2; 4; 6; 0|];
 [|8; 5; 9; 3|]|]
```

On considèrera `sol.(i).(j)` qui vaut la plus grande somme de nombres selon un chemin dans le sous-triangle de sommet (i, j) (dans la représentation dans une matrice). Par exemple le sous-triangle de sommet $(1, 0)$ est :

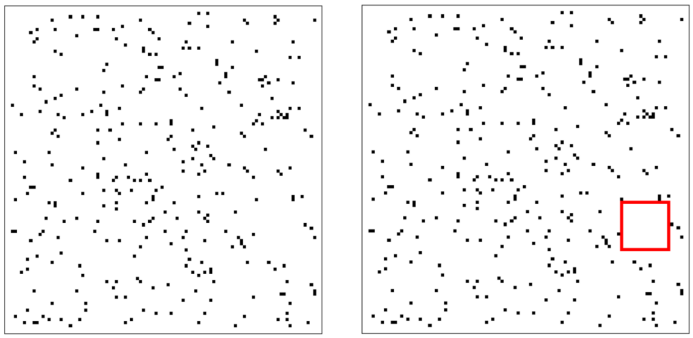
```
  7
 2 4
8 5 9
```

Trouver l'initialisation et une formule de récurrence pour `sol` et programmer l'approche descendante (on pourra utiliser une matrice pour représenter `sol`).

Exercice 3 : Plus grand carré blanc dans une image (en Ocaml)

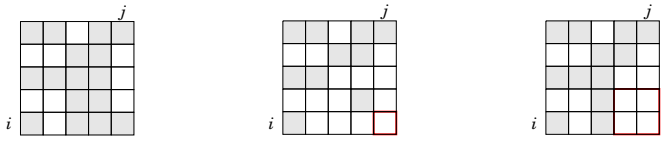
Dans cet exercice on considère une image de $n \times p$ pixels noirs ou blancs. Cette image est modélisée par un tableau `im` de taille $n \times p$ qui ne contient que des 1 (pixel blanc) et des 0 (pixel noir). Attention, la case (0,0) de la matrice est le coin supérieur gauche de l'image.

On cherche dans l'image le plus grand carré constitué uniquement de pixels blancs. Pour l'exemple suivant, il s'agit du carré marqué en rouge.



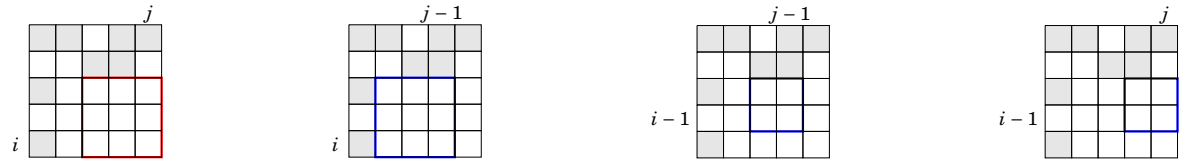
On note $tc(i,j)$ la taille du plus grand carré blanc qui se termine en (i,j) c'est à dire dont le coin inférieur droit inclut la case (i,j).

Pour les exemples suivants on a :



$tc(i,j) = 0$ $tc(i,j) = 1$ $tc(i,j) = 2$

La sous structure optimale vient du fait que la valeur de $tc(i,j)$ dépend de la valeur de tc en les 3 pixels qui sont au-dessus ou à droite du pixel (i,j). On observe que $tc(i,j) = 3$ car :



On a $tc(i,j) = 3$ car $tc(i,j-1) = 3$ $tc(i-1,j-1) = 2$ $tc(i-1,j) = 2$

On va en déduire une relation de récurrence :

- Si le pixel (i,j) est noir, alors $tc(i,j) =$,
- Sinon, le pixel (i,j) est blanc et la valeur de $tc(i,j)$ dépend des valeurs de tc pour les trois pixels à gauche ou au dessus de (i,j) :

$$tc(i,j) =$$

- **Q1.** Écrire la relation permettant de calculer $tc(i,j)$.
- **Q2.** Écrire une fonction `tableau_tc : int array array -> int array array` qui crée, remplit et renvoie le tableau tc à partir de l'image `im`, selon une approche ascendante.
  En vertu de la sous-structure optimale trouvée, le remplissage du tableau doit se faire de gauche à droite et du haut vers le bas. On pensera à initialiser correctement la première colonne et la première ligne.
- **Q3.** Écrire une fonction `plus_grand_carre : int array array -> int*int*int` qui prend en argument une image sous forme d'une matrice de valeurs 0 ou 1 et qui renvoie (i,j,r) où le plus grand carré blanc dans l'image est de taille r et son coin inférieur droit est en (i,j).
- **Q4.** Tester sur de petits exemples.
- **Q5.** Écrire une fonction qui modifie une image en plaçant la valeur entière 2 sur toutes les cases d'un plus grand carré blanc trouvé dans l'image. (Il n'y a pas unicité de la solution)
- **Q6.** Afficher le résultat dans le terminal en utilisant la fonction d'affichage pré-écrite.

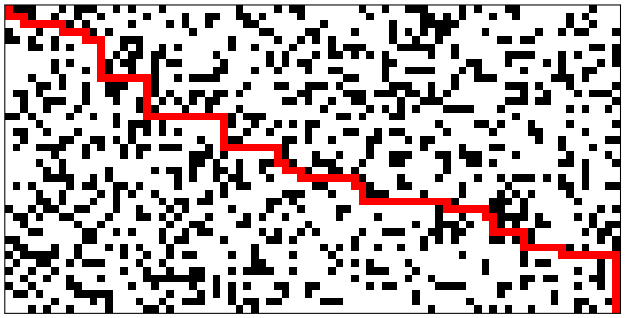
Exercice 4 : Chemin dans une grille (en C)

On considère une grille de taille $n \times p$ à cases noires et blanches, représentée par une matrice de taille $n \times p$ dont les valeurs sont 0 ou 1.

On veut déterminer un plus court chemin du coin inférieur droit $(n-1, p-1)$ vers le coin supérieur gauche $(0,0)$ qui ne passe que par des cases blanches en se déplaçant uniquement vers la gauche $(i,j) \to (i-1,j)$ ou vers le haut $(i,j) \to (i,j-1)$.

Remarque : On peut montrer que si un tel chemin existe alors il est toujours de taille optimale, donc notre problème revient aussi à juste chercher un tel chemin

Par exemple, pour la grille suivante un chemin est :



On va remplir un tableau T tel que $T[i][j]$ vaut la longueur d'un plus court chemin allant de (i, j) à la case $(0, 0)$ ou 2^{32} (l'infini) si la case (i, j) n'est pas accessible par un chemin depuis $(0, 0)$. On remarquera que le chemin n'a aucune raison d'être unique.

- **Q1.** Déterminer une formule de récurrence pour T.
- **Q2.** Écrire une fonction `calcul_T` qui remplit le tableau T ligne par ligne et colonne par colonne puis le renvoie.
- **Q3.** Vérifier sur les grilles d'exemples celles qui possèdent un chemin entre leurs coins en bas à droite et en haut à gauche.
- **Q4.** On peut reconstituer un chemin en utilisant T.

  En partant de la position (i, j) la plus en bas à droite, on remonte soit vers la gauche en $(i-1, j)$ si $T(i-1, j) = T(i, j) - 1$ soit vers la droite en $(i, j-1)$ si $T(i, j-1) = T(i, j) - 1$. Stocker les coordonnées des cases par lesquelles le chemin passe dans une liste `chemin`.

  Dessiner le chemin sur la grille en mettant un 2 là où le chemin passe. Vous pouvez utiliser la fonction d'affichage pour voir le chemin.