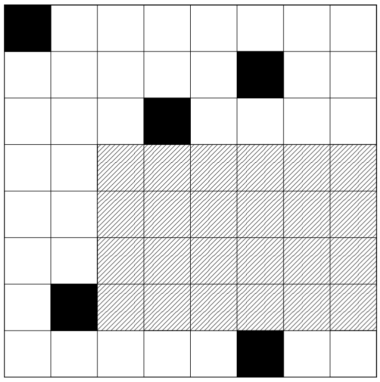
Rectangle d'aire maximale dans une grille de mots croisés

Etant donnée une grille de mots croisés (vide, c'est-à-dire une grille dont certaines cases sont noires et les autres blanches), on se propose de déterminer le rectangle d'aire maximale formé de cases blanches. Exemple :



Initialisation

On considère une grille carrée de $N \times N$ cases et soit k le nombre de cases noires dans la grille. On représente la grille par un tableau bidimensionnel `grille` de booléens, la valeur `true` représentant une case noire. On prendra $N = 10$ et $k = 6$.

□ Ecrire une fonction `raz : unit -> unit` qui met toutes les cases du tableau `grille` à la valeur `false` et une fonction `init : unit -> unit` qui noircit aléatoirement *exactement k cases* du tableau `grille`. (On obtient un entier aléatoire entre 0 et $n-1$ par `random_int` n).

Affichage

On souhaite représenter graphiquement la grille et la solution du problème. Rappel : pour utiliser la fenêtre graphique de Caml Light, on ouvre le module `graphics` puis on appelle la fonction `open_graph`.

```
#open "graphics";;
open_graph "";;
```

La taille de la fenêtre graphique est obtenue avec `size_x()` et `size_y()`.

□ Ecrire une fonction `affiche_case : int -> int -> unit` qui affiche la case (i, j) de la grille (vide si le booléen correspondant est faux, et pleine sinon). En déduire une fonction `affiche_grille : unit -> unit` qui affiche la grille entière.

Un intervalle d'entiers consécutifs $i, i+1, \ldots, i+d-1$, commençant à l'entier i et de longueur d, sera représenté par le couple (i, d) et un rectangle de la grille par deux ses les deux intervalles qu'il représente sur les abscisses et les ordonnées.

```
type intervalle = Interv of int * int;;
type rectangle  = Rect   of intervalle * intervalle;;
```

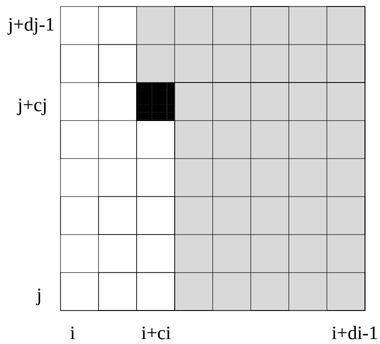
□ Ecrire une fonction `affiche_rectangle : rectangle -> unit` qui affiche, dans une autre couleur que la grille, un rectangle passé en argument.

Solution au problème

La solution qui consiste à passer en revue tous les rectangles de la grille, à tester s'ils sont composés de cases blanches uniquement et à conserver celui d'aire maximale n'est pas satisfaisante car trop coûteuse. Quelle est, d'ailleurs, la complexité de cette méthode ?

On propose ici une autre solution, plus efficace. L'idée est de remarquer que, pour chaque case noire en position (x, y) dans la grille, alors la solution se trouve dans l'une des quatre portions de la grille définies par $i < x$, $i > x$, $j < y$ ou $j > y$ (en effet, une solution ne peut contenir la case (x, y)).

Soit alors une région $\{i, i+1, \ldots, i+di-1\} \times \{j, j+1, \ldots, j+dj-1\}$ dans laquelle on recherche une solution. Cette solution est dont représentée par les deux intervalles `Interv` (i,di) et `Interv` (j,dj). On commence par cherche la première colonne contenant une case noire, et dans cette colonne la case noire d'ordonnée la plus petite. Soit $(i + ci, j + cj)$ cette case noire. Exemple :



On rappelle alors récursivement la procédure sur les quatre portions définies ci-dessus et on conserve celle d'aire maximale. (Remarque : pour la portion de gauche, on connaît déjà le rectangle d'aire maximale par choix de la case noire).

□ Ecrire une fonction `case_noire : intervalle -> intervalle -> int*int` recherchant la case noire ci-dessus, donnée par (ci, cj), dans une portion de la grille définie par deux intervalles. En déduire une fonction `trouve : unit -> rectangle` qui donne la solution du problème. Visualiser la grille et sa solution.