État de l'art

Dimitri Granger   Matthias Goffette

*Nous avons cherché à modéliser un réseau décentralisé pair-à-pair (*P2P*), qu'aucun utilisateur ne quitte ou ne rejoint.*

1 Positionnement thématique et mots-clefs

2 Bibliographie commentée

3 Problématique retenue

Plusieurs problématiques s'offrent à nous :

- **Comment rechercher efficacement une information sur un réseau ? Application : Moteur de recherche décentralisé.**
  - Comment s'assurer que les informations reçues sont *de confiance* ?
- (Comment organiser le partage de fichiers efficacement (cas d'un iso *linux*) ? Rajouter des contraintes : combien de noeuds maximum ?)

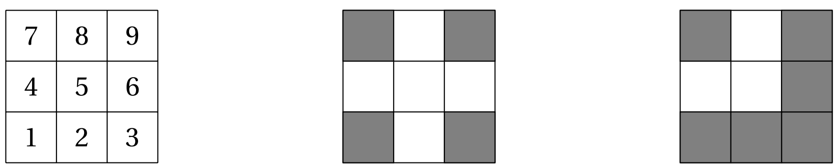
4 Objectifs du travail

5 Liste des références bibliographiques

6 Reprise d'un document précédent

6.1 Paragraphe 1

On peut représenter une case du labyrinthe par un carré de 3×3 pixels (1a) : la case elle-même se trouve au pixel 5. Les pixels 2,4,6 et 8 servent à faire le lien entre la case et les cases adjacentes. Un tableau de $(2 * n) + 1$ pixels est nécessaire. Les pixels 1,3,7 et 9 sont alors toujours noirs, le pixel 5 toujours blanc, et les pixels 2,4,6 et 8 sont noirs ou blancs, selon la possibitité de déplacement d'une case à l'autre (1b). Par exemple, à partir de la case représentée en 1c, on peut se déplacer vers le haut ou vers la gauche, mais pas vers le bas ou vers la droite.



(a) une case (b) case remplie (c) exemple

Figure 1: Représentation d'une case

6.1.1 Dessin du labyrinthe

Nous allons réaliser l'enregistrement du labyrinthe au format PGM. Ce format permet d'enegistrer facilement des images en nivau de gris. Définissons donc les couleurs des pixels :

En premier lieu, il faut compléter la fonction labyrinthe, en ajoutant un tableau `lislab`, de dimension $(2n+1)\times(2n+1)$ qui contiendra la représentation du labyrinthe. Le tableau étant par défaut tout noir, on commence par le préremplir en blanchissant les pixels représentant les cases.

Il faut ensuite lier les cases entre elles lors de la création du labyrinthe. On remarque de manière astucieuse, que pour deux cases (x,y) et (x',y') adjacentes, le pixel entre les pixels les représentant a pour coordonnées $(x+x'+1, y+y'+1)$. Le code suivant permet donc de faire ce lien.

Il suffit alors de retourner le tableau de pixels à la fin de la fonction : la fonction `save_lab` se contente ensuite de sauvegarder le labyrinthe au format désiré.

6.2 Obtention du chemin

6.2.1 Une remarque astucieuse

La question suivante consiste à construire à la volée le chemin reliant les cases $(0,0)$ et $(n-1,n-1)$. Le point clé de cette question réside dans les trois lignes suivantes : On se rend alors compte qu'on n'ajoute jamais à la pile qu'une case adjacente à la case du sommet de la pile. Ainsi, si l'état de la pile est 2a, alors les cases C_{k-1} et C_k sont adjacentes ; le chemin de C_{k-1} à C_k est alors $[C_{k-1}, C_k]$. Par récurrence, si la pile est comme représenté en 2b, alors le chemin allant de C_0 à C_n est exactement l'ensemble des cases contenues dans la pile. Or C_0 est la case $(0,0)$. Ainsi, en appliquant ce résultat pour $C_n=(n-1,n-1)$, si `cellule` contient $(n-1,n-1)$, alors `pile` contient le chemin allant de $(0,0)$ à $(n-1,n-1)$.

$\cdots$
C_{k-1}
C_k
$\cdots$
C_0

(a) Pile k

C_n
C_{n-1}
$\cdots$
C_1
C_0

(b) Pile à l'arrivée

Figure 2: divers états de la pile

6.2.2 Implantation

Il s'agit donc simplement de faire une copie de la pile, sans modifier cette dernière, quand le sommet de la pile vaut $(n-1,n-1)$. Le code suivant permet de copier une version renversée de la pile. Le principe est simple : en appelant `revdup(p)`, on dépile les éléments de `p` pour les empiler dans deux piles : on utilise l'une pour reconstituer p, et on retourne l'autre.

On a donc besoin d'une pile `chemin`, qui contiendra le chemin à la fin. Il suffit ensuite de recopier la pile dans `chemin` au bon moment, puis de tracer le chemin dans le tableau de pixels :

Figure 3: le labyrinthe