FERNANDEZ
Thomas
PI2

Projet Programmation : Compte rendu de la première séance.

0. Structures utilisées.

Un pixel sera représenté par ses coordonnées dans la matrice 2D qui représente le réseau.

```
struct pxl {
        int x;
        int y;
} ; typedef struct pxl pixel;
```

Afin de pouvoir utiliser la structure de données de type « pile » (cf I.B. fonction int* recherche(reseau)), il est également nécessaire d'implémenter la structure :

```
struct stack {
        int tete;
        PIXEL* liste;
} ; typedef struct stack PILE;
```

Le sujet que j'ai décidé de traiter s'intéresse à la théorie de la percolation. Sa réalisation peut-être décomposée en 5 grandes étapes.

I. Recherche des chemins de percolation.

L'objectif de cette première partie est de réaliser une partie de programme capable de charger d'une image existante ou de générer aléatoirement un réseau puis de rechercher s'il existe un chemin de percolation à l'intérieur.

A. Obtention du réseau.

- *reseau.h* :
    - int ** creationReseau(int x, int y);
    - int** reseauAleatoire(int x, int y, float p);
    - int* reseauLu(char* image);

- *reseau.c :*
    - int** creationReseau(int x, int y){
        // Cette fonction crée une matrice 2D formée contiguëment de x colonnes et de y lignes.
}
    - int** reseauAleatoire(int x, int y, float p){
        // Cette fonction renvoie un nouveau réseau aléatoire de x colonnes et de y lignes dans laquelle chaque noeud a une probabilité p d'être à l'état ON (valeur 1).}

- int* reseauLu(image){
// Cette fonction lit une image PGM binaire et renvoie un réseau sous la forme d'une matrice 2D.}

B. Recherche d'un chemin de percolation dans un réseau.

La recherche récursive de percolation se présente comme une solution simple pour résoudre le problème souhaité mais n'est pas applicable dans le cadre d'images de grandes tailles. Or, plus la taille des images augmente, plus les résultats seront justes. C'est pour cela que ce sera la méthode itérative qui sera utilisé par la suite.

- *recherche.h :*
  - int rechercheParPxl(pixel pxl);
  - int rechercheRecursive(int** reseau);
  - int* recherche(int* reseau);

- *recherche.c :*
  - int rechercheParPxl(pixel pxl){
// Cette fonction cherche récursivement s'il existe un chemin de percolation en partant d'un pixel pxl et en étudiant ses voisins.}

  - int rechercheRecursive(int** reseau){
// Cette fonction parcourt l'intégralité des pixels qui composent la première ligne de l'image afin qu'ils servent de points de départ pour la fonction rechercheParPxl dans l'attente de trouver un chemin de percolation. Elle renvoie 1 si la recherche s'avère fructueuse, 0 sinon.}

  - int* recherche(reseau){
// Cette fonction recherche l'existence d'un chemin de percolation dans le réseau reseau. Elle renvoie une copie de ce réseau dont tous les pixels conducteurs étudiés sont distinguables (valeur 2) si la recherche s'avère fructueuse, 0 sinon. En comparant les valeurs des pointeurs en entrée et en sortie, il est possible de simplement savoir si la recherche a été fructueuse ou non.}

II. Evaluation de la probabilité qu'un réseau percale en fonction de p.

- *evaluation.h :*
  - int* eval(int nbtests);
  - void stockageCSV(int* resultats);
  - void traceGraphe();

- *evaluation.c :*
  - int* eval(nbtests){
// Cette fonction prend en entrée le nombre de tests à réaliser pour chaque probabilité p puis réalise les tests demandés, calcul la probabilité qu'un

réseau percale en fonction de p et résume les résultats dans un tableau de 2 colonnes proche du format CSV.}

- void stockageCSV(resultats){
  // Cette fonction prend en entrée un tableau d'entiers, résultat de la fonction eval et produit le fichier CSV correspondant.}

- void traceGraphe(){
  // Trace le graphe représentant l'évolution de la probabilité qu'un réseau percole en fonction de p}

III. Visualisation du problème.

- *visualisation.h :*
  - void visuPGM(int* reseau);
  - void grossissement(int G, char*¨ image);
  - void visuGIF(int** ecoulement);

- *visualisation.c :*
  - void visuPGM(reseau){
    // Cette fonction prend un réseau en entrée et génère l'image PGM binaire correspondante}

  - void grossissement(G, image){
    // Cette fonction génère un double de l'image en entrée, grossis d'un facteur G}

  - void visuGIF(ecoulement){
    // Cette fonction prend en entrée un tableau contenant l'état d'un réseau après chacune des nouvelles affectations d'un pixel à l'état « conducteur » et produit un GIF animé retraçant l'exploration du potentiel chemin de percolation.}

IV. Divers.

Cette partie contiendra la parallélisation du problème, ainsi que l'adaptation à d'autres types de réseaux.