Rapport de projet Picross

CONNES Victor, saississez vos noms

April 21, 2015

Contents

1 Description de l'implémentation

1.1 Les besoins du programme :

1.1.1 Pour les indices de chaque ligne/colonne:

- Imperatif à satisfaire:
  - structure de taille dynamique (nombre variable dindice par ligne)
  - parcours en θ(n) (plusieurs méthodes recourant à un parcours total ou partiel de la structure)
  - relation dordre (le parcours ne sont réaliser que dans lordre de lindice le plus en haut vers celui le plus en bas respectivement gauche-droite pour les lignes)
- Choix : la liste simplement chainés, car elle correspond parfaitement au spécification et parait finalement très proche de la réalité

1.1.2 Pour représenter la grille du picross :

- Imperatif à satisfaire:
  - structure de taille fixe
  - accès en temps constant a chaque case
  - faciliter de copie de la structure (θ(n))
- Choix: La matrice car très proche de la réalité

1.1.3 Pour lensemble des indices de lignes/colonnes :

- Imperatif à satisfaire:
  - accéder en temps constant a chaque liste
  - garder un indicage cohérent avec la matrice
- Choix :Un tableau

1.1.4 Pour ligmodif /colmodif

- Imperatif à satisfaire:
  - ajout dun élément en temps constant
  - retrait dun élément en temps constant
  - taille dynamique
- Choix : Le choix naturel serait lensemble mais on choisit la liste simplement chainés car déjà implémentés pour les indices de chaque ligne/colonne. Son comportement est similaire dans le cas d'un ajout et d'un retrait en début de liste .

1.2 Les classes

L'ensemble de nos classes disposent d'une surcharge de l'operateur¡¡ afin de faciliter l'affichage dans le terminal. De plus, il existe d'autres classes dans notre programme qui permettent d'améliorer l'affichage en utilisant une interface graphique. Nous ne distinguerons ici seulement les classes permettant la resolution du picross.

1.2.1 La classe Cell :

Elle représente un indice logique

- Attributs :
  - Val : qui représente la valeur de cette indice logique (cette valeur étant strictement entière positive et a priori non borné nous avons choisit de la représenter par un size_t)
  - Suiv : qui est pointeur sur la cellule suivante de la liste

1.2.2 La classe Liste :

- Attributs :
  - Longueur : qui représente la longueur de la liste. Cette attribut et incrémenter ou décrémenter automatiquement dès qu'il y a variations de la taille de la liste.
  - Fini : Nous indique si la ligne/colonne a laquelle est rattaché la liste est entièrement rempli (booléen à true) ou non (booléen à false), cette attribut et notamment très utile pour vérifier la condition d'arrt de notre résolution (cf. main)
  - Tte : Pointeur vers le premier élément de la liste.
- Méthode remarquable :
  - Surcharge de l'opérateur() : Nous avons utiliser l'operateur() comme un operateur d'indexage d'une liste cela nous permet notamment de faciliter le parcours d'une liste.

1.2.3 La classe Tabliste :

- Attributs :
  - Tab : le tableau de liste
  - Taille : la taille du tableau
- Méthode remarquable :
  - Surcharge de l'opérateur[]: Nous permet d'utiliser comme si elle était simplement un tableau.

1.2.4 La classe Matrice :

- Attributs:
  - Tab : la matrice d'entier (-1 : blanc 0 : indéterminé 1 : noir)
  - Nbl : Nombre de ligne
  - Nbc : Nombre de colonnes

1.3 La classe Picross :

La classe picross est la classe dont les instances représente les picross c'est dans cette classe que sont implémenté nos méthodes de résolution.

1.4 Les algorithmes remarquables :

1.4.1 SLG :

Spécification : void SLG(int*, size_t, Cell*, size_t, bool&);

- Paramètre:
  - Un tableau d'entier: (Done/resultat) il représente une ligne/colonne de notre matrice, initialement il peut contenir des cellules à 1,-1,0
  - Un size_t : représentant la taille du tableau
  - Une cellule de liste: il repésente le prochain indice a placer dans le tableau, initialement la première cellule associé à la dites ligne/colonnes
  - Un size_t : représentant l'indice dans le tableau auquel on souhaite placer le bloc, initialement 0
  - Un booléen pass par référence : reprsentant si il est possible ou non de placer la fin de la liste d'indices indice courant compté a partir de l'indice courant dans le tableau.
- Retour de la fonction.

Le retour ce fait par lintermédiaire du tableau et du booléen. Si le booléen est à false c'est que le couple (ligne/colonne,liste d'indice) n'a pas de solution.Dans le cas contraire, la solution la plus à gauche nous est don´e dans le tableau.

Complexité pire des cas : $\theta(n^2)$
Notons :

- i: l'indice dans notre tableau
- n: la taille du dit tableau
- T: le dit tableau
- m: la taille de l'indice courant dans la liste

Dans le pire des cas de notre algorythme, on peut imaginer que chaque hypothése faites par notre algorythme soit fausse notre fonction serait donc appeler: n-(i+1) á chaque case en effet,la premiére case entrainerait n-1 appel, la deuxieme n-2 ainsi de suite. Sans compter l'appel récursif notre algorythme à un coup de m pour placer un bloc mais comme pas definition m¡n on considerera son coup á n. Ainsi on obtient l'équation recursive suivant:
n + SLG(i+1)
Soit une complexité inférieure à n^2.

Complexité meilleur des cas : θ(n)