clement.rouvroy@ens.psl.eu

Les liens dansants

Prélude Ce sujet est plus compliqué que la moyenne (vraiment), il est fait pour vous apprendre à composer sur des sujets qui sortent de votre routine. N'abandonnez pas, cherchez à faire le plus possible, la première partie sert de mise en bouche, la traiter sur 2 heures ne pourra pas octroyer une bonne note, il faut impérativement attaquer la suite du sujet.

Thèmes Retour sur trace, pointeurs **Type** X/ENS (très théorique)

Consignes Le candidat respectera le langage imposé (C) et ne devra pas utiliser de librairie hors-programme. Il est invité à **faire des schémas clairs et précis** de ses algorithmes lors des appels au correcteur, c'est à la fois un gain de temps pour les deux et une manière de montrer qu'il a compris ce qu'il manipule. Les preuves écrites doivent être formelles, sauf si la consigne précise que ce n'est pas nécessaire, la rigueur sera évaluée. L'examinateur ne débugera pas votre code, en revanche en cas de doute ("ai-je le droit à telle ou telle librairie ?", "je suis sortie de la VM, comment y revenir ?", "ai-je le droit à une indication pour cette question ?") n'hésitez pas à poser votre question. Elle ne vous dévalorisera pas, si la réponse peut vous retirer des points, l'examinateur vous demandera avant de vous donner la réponse si vous l'acceptez (*si vous n'avez pas de chance, le jour de l'oral il vous retirera les points rien que pour avoir posé la question, par exemple si vous demandez "comment trier une liste en $O(n\,log(n))$?" ou un autre résultat classique du programme, ça sera sûrement retenu contre vous*). Enfin, si l'examinateur n'est pas à votre portée quand vous avez besoin d'une aide / d'une question oral à donner, gardez le bras levé et lisez la suite, ne restez jamais passif.

Introduction du sujet

Dans ce TP, vous allez apprendre à utiliser l'algorithme **X** - que l'on doit à un article de recherche de Donald E. Knuth, auteur des **The Art Of Computer Programming**-. Il utilise une strucutre de donnée ingénieuse permettant d'accélérer certains algorithmes initialement lents à cause de la nature compliquée des problèmes triatés (on parle de problème NP-Complet, mais ce sera vu au programme de spé)

I - Liste doublement chaînée

Avant d'attaquer le vif du sujet, vous allez devoir implémenter une structure de donnée usuelle : les listes doublement chaînées. Chaque élément (ecepté le premier et le dernier) a accès à l'élément à sa gauche (prédécesseur) et l'élément à sa droite (successeur).

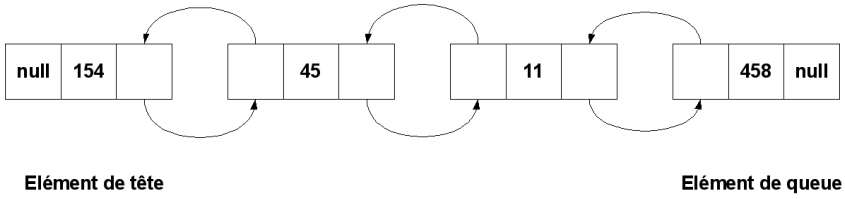

Figure 1: Une liste doublement chaînée (Wikipedia)

Une liste doublement chaînée est dîte **circulaire** si on ajoute les deux règles suivantes:

- le successeur du dernier élément est le premier élément
- le prédécesseur du premier élément est le dernier élément

Dans le reste du sujet (convention sans importance mais qu'il faut fixer), quand on ajoute un élément dans une liste circulaire, on l'ajoute au début de la liste (au premier élément).

Question 1 code

Créez un fichier `liste.h` et mettez-y le code suivant :

```
struct node{
int val;
struct node* left;
struct node* right;
};
?? // Ecrire une ligne permettant de faire appelle au type node et non au
type struct node.
?? // Ecrire une ligne permettant de faire appelle au type liste au lieu
de
node*
liste creer_liste_vide();
void insert(liste l, int x);
void remove(liste l, int x);
void free_liste(liste l);
```

Implémentez ces fonctions dans un fichier `liste.c`

II - Le problème de couverture exacte

N'utilisez pas de liste doublement chaînée dans cette partie, si vous connaissez déjà les liens dansants *(bravo)* : abstenez-vous !

Definition 1 Problème de couverture exacte

Entrée Une matrice carrée avec uniquement des 0 et des 1

Sortie Peut-on prendre un ensemble de lignes de manières à avoir un seul et unique 1 sur chaque ligne ? Si oui, la donner.

Question 2 dans votre tête

Répondre au problème de couveture exacte sur l'entrée suivante:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Question 3 à l'écrit

Définir le **retour sur trace**.

Donner une stratégie de retour sur trace permettant de résoudre ce problème. Maximum 10 lignes, soyez le plus précis possible sur les points qui demandent de la précision et le plus général possible ailleurs (ne perdez pas de temps ni de ligne).

III - L'Algorithme X et les liens dansants

III.1 - Introduction à l'algorithme X

L'algorithme X a été introduit par Donald E. Knuth (dans un papier publié en Novembre 2000, à Stanford) et est juste une formalisation du backtracking avec **une** spécificité. Voici l'algorithme:

```
X(A):
1  if A = ∅:
2      fin
3  Choisir une colonne c
4  Choisir une ligne r (non-déterministe), telle que A[r, c] = 1
5  Inclure r dans la solution partielle
6  for j tel que A[r, j] = 1:
7      retirer la colonne j de A
8      for i tel que A[i, j] = 1:
9          retirer la ligne i de A
10 X(A)
```

Il y a deux nuances dans cet algorithme :

- Le choix de c est déterministe (il est codé selon une stratégie précise qu'on a choisie)
- Le choix de r est non-déterministe, c'est une notion d'informatique fondamentale qui signifie qu'on peut imaginer que le programme se duplique pour tester toutes les possibilités possibles (sur l'arbre des choix il choisie une branche)

Pour le premier cas, on connaît plusieurs stratégies et ce ne sera pas vraiment un problème (vous pourrez en tester plusieurs si vous êtes assez rapide).

Pour le second, le non-déterminisme n'est pas très utile en code car on ne peut pas lefaire (ou alors on testerait toutes les possibilités à chaque étage de la récurrence, c'est peine perdue tout comme la solution par retour sur trace). On va donc utiliser les **liens dansants** pour retirer cette partie non-déterministe.

III.2 - Introduction aux liens dansants

On note $L[N]$ la liaison gauche (Left) du noeud N et $R[N]$ la liaison droite (Right) du noeud N.

Question 4 à l'écrit

Soit x un neoud qui a un élément non nul à gauche et un élément non nul à droite. Que fait le combo d'opérations suivant ?

$$L[R[x]] \leftarrow L[x], R[L[x]] \leftarrow R[x]$$

Proposer un combo de deux opérations qui annule ces changements.

C'est ce que Knuth appelle les **liens dansants**, oui oui, ce concept aussi simple peut réellement être utile !

III.3 - Utilisation des liens dansants

On a désormais toutes les clés en mains pour utiliser les liens dansants : un type et un problème. Voici les instructions détaillés pour utiliser les liens dansants :

Le problème avec la structure de matrice classique est que la recherche d'un 1 et des colonnes spécifiques prend trop de temps, on aimerait pouvoir faire cette recherche sans chercher à chaque fois, l'idée va être de **lier les 1 entre eux** et de définir une méthode de déplacement dans cette structure. On va coder une matrice A en un objet x de cette manière:

- Les lignes sont reliées par deux listes doublement chaînées circulaires (LDCC) $L[x]$ et $R[x]$ (Left et Right)
- Les colonnes sont reliées par deux LDCC $U[x]$ et $D[x]$ (Up et Down)
- Chaque colonne a un objet spécial y qui contient:
  - $L[y]$ et $R[y]$ (les colonnes à gauche et droite qui n'ont pas encore été prises, ainsi qu'une colonne spéciale H)
  - H est une colonne spéciale appellée la racine, on en reparle dans l'algortihme
  - $U[y]$ et $D[y]$ (le dernier et le premier 1 de la colonne)
  - $S[y]$ (la taille de y, son nombre de 1)
  - $N[y]$ (son nom, pour l'affichage)

Par exemple, la matrice de la question 2 ressemble à ceci avec cette structure:

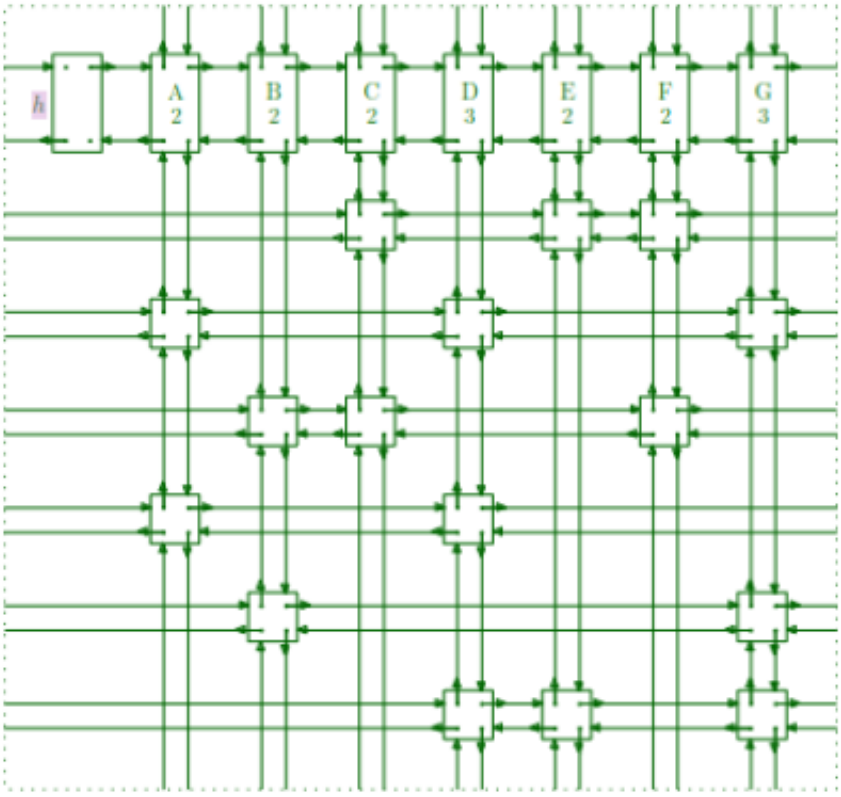

Figure 2: provient de l'article de Knuth

Question 5 code

Proposer un type permettant de stocker l'objet spécial de chaque colonne (on l'appellera `header`, en accord avec le papier de D. Knuth).

Question 6 code

Définir un type permettant de stocker une matrice sous forme de lien, on l'appellera `struct dansant` et on lui donnera l'alias `dansant`

Question 7 code

Ecrire une fonction `dansant matrice_vers_dansant(int** matrice)` qui prend une matrice `0-1` et renvoie la structure de lien dansant.

III.4 - Cassage du non-déterminisme

On peut désormais réécrire l'algorithme **X** sans non-déterminisme avec la stratégie de recherche suivante:

```
RECHERCHE(k):
 1  if R[h] = h:
 2      ?
 3  c ← R[h]
 4  couvrir(c) for each r ← D[c], D[D[c]], ... while r ≠ c:
 5      O_k ← r
 6      for each j ← R[r], R[R[r]], ... while j ≠ r:
 7          couvrir(j)
 8      recherche(k + 1)
 9      r ← O_k
10      c ← C[r]
11      for each j ← L[r], L[L[r]], ... while j ≠ r:
12          decouvrir(j)
13      decouvrir(c)
```

Vous ne devez probablement plus avoir beaucoup de temps donc les questions vont vous êtres plus mâchés.

L'opération de couverture doit **retirer c de la liste des Headers** (que vous avez définie précédemment) et retirer toutes les autres colonnes les 1 de c dans leur listes. En voici un exemple:

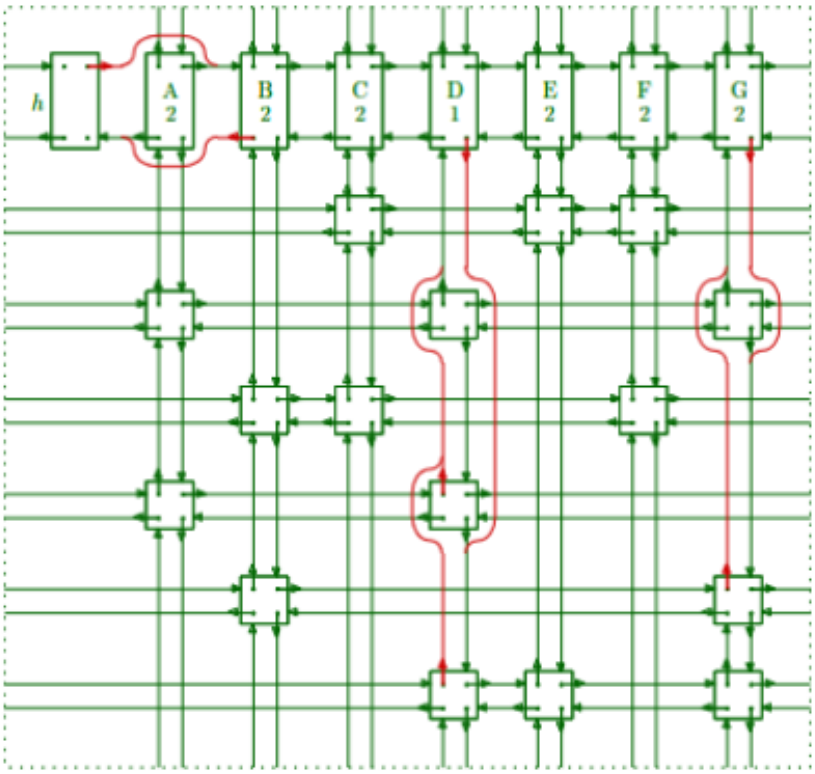

Figure 3: Couverture de A

Question 8 à l'écrit

Que faire dans le cas $R[h] = h$?

Question 9 à l'écrit

Donner le pseudo-code de l'opération couvrir (ou le code si vous pensez que c'est plus rapide, mais seulement dans ce cas !)

Remarque: Indication : La première opération des liens dansants peut être utiles

Question 10 code

Donner le pseudo-code de la fonction `decouvrir` qui annule `couvrir` en utilisant les **liens dansants** (ie passer de la figure 3 à la figure 2)

Question 11 code

Implémenter ces opérations.