Solution de l'épreuve finale de complexité

Exercice 1 :

L'algorithme peut être inspiré du parcours en profondeur ou directement à partir de l'algorithme de parcours en largeur.

Barème détaillé (Algorithme : 2pts principe, 2pts parcours, 2 pts pour la création du tableau sans écrasement)

```
Parcours(racine : arbre, i : entier) ;
Début
Si (racine <> nil) alors T[i] :=racine->val ;
Parcours(racine->fg,2*i) ;
Parcours(racine->fd, 2*i+1) ;
Fsi ;
Fin.
```

```
Parcours_prefix(racine :arbre)
Début
I:=1; A:=Racine; Afficher(A.clé);  Empiler(A,P);
empiler(i) ; T[i] :=A.clé ;I := 2*i; A :=A->fg ;
Tant que (non Vide(P)) faire
        Tant que (A <> null) Faire
        Empiler(A,P); empiler(i) ; T[i] :=A.clé ;
        A:=A->fg; i :=2*i ;
        Fait;
        J :=Depiler(P) ; A:=Depiler(P);
        tant que ((tete(P)->fd = A) et non_vide(P))
        faire
                J :=Depiler(P) ; A:=Depiler(P);
                T[j+1] :=A.clé ;
        fait;
        Si (non_vide(P))
                alors i :=Depiler(P)+1 ;
                        A := Tete(P)->fd;
/* sommet frères ou bien un encêtre fd*/
                Sinon A:= null;
                Fsi;
        Fait;
Fait;
Fin.
```

Exercice 2 :

Questions :

1. Résoudre le problème des n-Reines
    a. Décrire clairement les structures de données utilisées et le traitement le plus adéquat à effectuer pour résoudre le problème des n-Reines (3 pts)

Les structures de données : deux modélisations possibles pour ce problème : (1pt)

- Soit une matrice M de dimension n*n.

$$M[i,j] = \begin{cases} 0 & si\ la\ case\ est\ libre \\ 1 & Si\ la\ case\ est\ occupé\ par\ une\ reine \\ -1 & Si\ la\ case\ est\ menacé\ par\ au\ moins\ une\ autre\ reine \end{cases}$$

- Soit un tableau de dimension n, tel que T[i] représente la position (la colonne) de la $i^{ème}$ reine dans la $i^{ème}$ ligne de l'échiquier. Une fonction est nécessaire dans ce cas pour déterminer si une case est menacée ou pas par rapport aux reines déjà placées.

Le traitement le plus adéquat à effectuer : (2 pts)

Initialement l'échiquier est vide (indépendamment de la structure adopté). Résoudre le problème consiste à placer les reines une par une on commençant par la première case de l'échiquier. Chaque fois qu'une reine est placée les contraintes du problème doivent être vérifiées et les cases menacées par ce nouveau placement doivent être mises à jour pour en tenir compte lors du placement des reines suivantes. Si pour une reine donnée (une ligne donnée i) aucune case n'est libre, le placement de la reine qui la précède (la ligne i-1) doit être remis en cause et déplacé à une autre position possible, et ainsi de suite jusqu'à ce que toute les reines puissent être correctement positionnées dans l'échiquier.

Ce traitement peut être concrétisé au moyen d'un algorithme récursif puisque le traitement proposé est par définition récursive. Néanmoins la version itérative n'est pas à écarter.

    b. Ecrire un algorithme permettant d'engendrer une solution au problème (3 pts)

```
fonction EST-LIBRE(S,rang,col):booléen; (1 pt)
DEBUT
  Libre :=vrai;
  r := 1;
  TQ (r ≤ rang) faire
    c := S[r];
          Si ((col = C) ou (abs(rang-r) = abs(col-c))) alors libre :=faux ;
                    Sinon r ← r + 1;
  Fait ;
  Retourner libre ;
FIN ;
```

Fonction placer_reine(entier reine) : entier ; (2 pts)

Début

```
 SI (reine > n) ALORS retourner(1) ;
  SINON
    col := 1;
    Tantque (col ≤ n) FAIRE
      SI EST-LIBRE(S,reine,col) ALORS
            S[reine] := col;
            Si (Placer_reine(reine + 1)=1)alors retourner(1) ;
            Sinon S[reine] :=0;
      FSI
      col := col + 1;
    fait ;
  FSI
Retourner(0) ;
FIN
```

Fin.

c. Calculer la complexité au pire cas de l'algorithme proposé (2 pts)

La complexité peut être calculée de deux manières différentes :

- La plus simple, c'est de ce dire que dans ce problème il est demandé de déterminer un emplacement pour n reines dans l'espace des n² cases de l'échiquier, ce qui correspond à un arrangement $A_n^{n^2}$, le nombre d'arrangement possible se définit par :

$$A_n^{n^2} = \frac{n^2!}{(n^2-n)!} = O(n!).$$

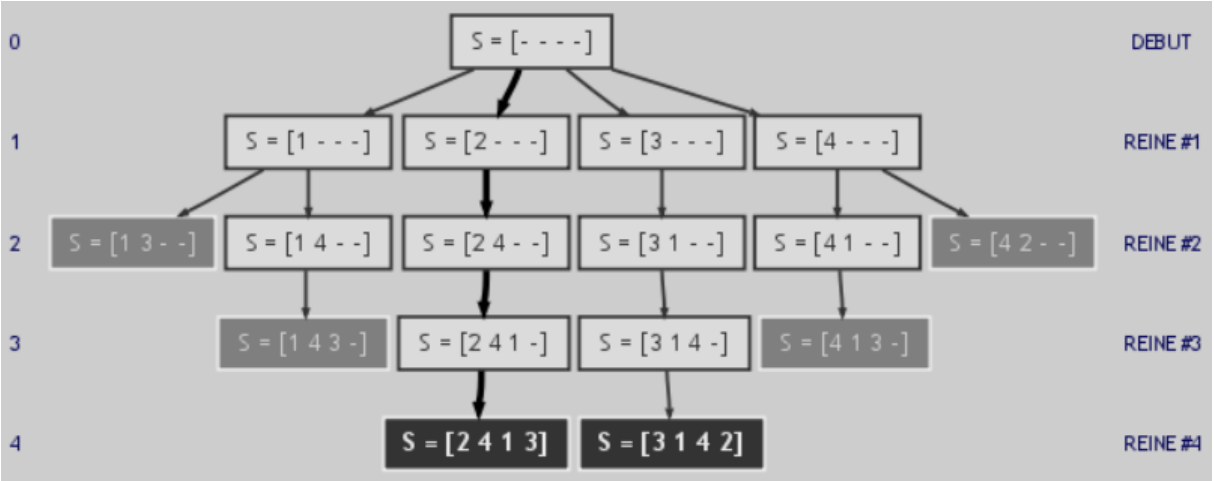
- Ou bien, en utilisant une équation de récurrence pour exprimer l'évolution du nombre d'itération au fil des appels récursifs. Au pire cas cette évolution peut être exprimée de la manière suivante :
- T(n) = O(n) + n * T (n-1). Avec T(n) = 1 si n =1.

La résolution de cette équation de récurrence nous donne T(n) <= n ! donc la complexité est de l'ordre de O (n !).

2. Illustrer votre proposition sur un échiquier 4*4. (3 pts)

Première modélisation :

Seconde modélisation :



Montrer que le problème des n-Reines appartient à la classe NP. (3 pts)

On dit qu'un problème donné A appartient à la classe NP s'il existe un algorithme de validation polynomial pour ce problème. Donc montrer que le problème des n-reines appartient à la classe NP revient à proposer un algorithme de validation et démontrer qu'il a une complexité d'ordre polynomial.

n_reine (solution s) (2 pts)

/* avec cette modélisation on est sure d'avoir une seule reine par ligne, il reste donc à vérifier par rapport aux colonnes et aux diagonales montante et descendante */

Début

j :=1 ;

Valide = vrai ;

```
Tant que ( j< n et valide = vrai)
Faire k :=1 ;
    Tant que (k<=n-i et valide = vrai) faire
        /* une seule reine par colonne */
        si S[j+k]=S[j] alors valide =faux ;
            /* une seule reine par diagonale*/
            sinon si S[j+k]=s[j]-k ou S[j+k]=s[j]+k alors valide = faux;
                sinon si (j-k) >0 alors si S[j-k]=s[j]-k ou S[j-k]=s[j]+k
                                        alors valide = faux ;
                                        sinon k:=k+1;
                                        fsi ;
                                fsi ;
            fsi ;
        fsi ;
    fait ;
```

La complexité de cet algorithme est de l'ordre de $O(n^2)$ donc le problème des n-reines appartient à la classe NP. (1pt)