Projet Backtracking

Remplissage d’une grille de Sudoku

Groupe : Aubin, Kyriann, Samy

Sommaire

- Présentation du problème
- Backtracking
  - présentation de la méthode
  - application au problème
- Présentation de code
  - types
  - fonctions
- Conclusion et perspectives

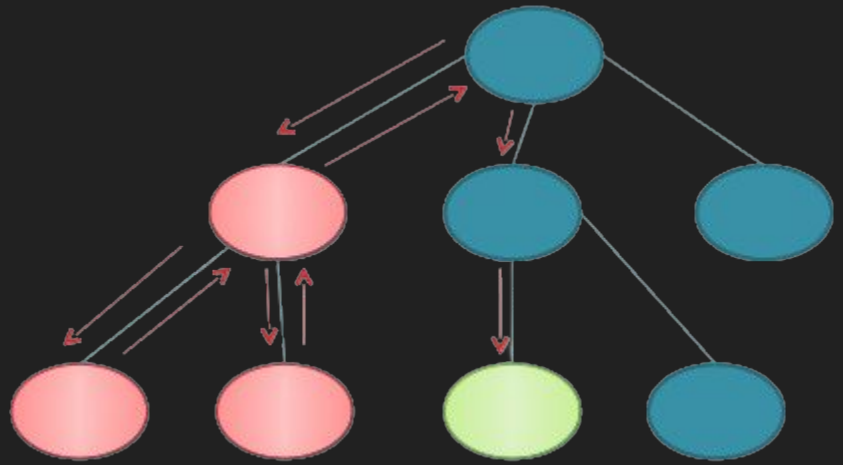
Présentation du problème

- Grille de Sudoku à remplir
- Modélisation : matrice 9x9
- Choix arbitraire d'**une** solution

9			1					5
		5		9		2		1
8				4				7
				8				
			7					
				2	6			9
2			3					6
			2			9		
		1	9		4	5	7	

Backtracking

- retour sur trace → recherche exhaustive (essais successifs)
- arbre de décision parcouru en profondeur (Depth-First search)

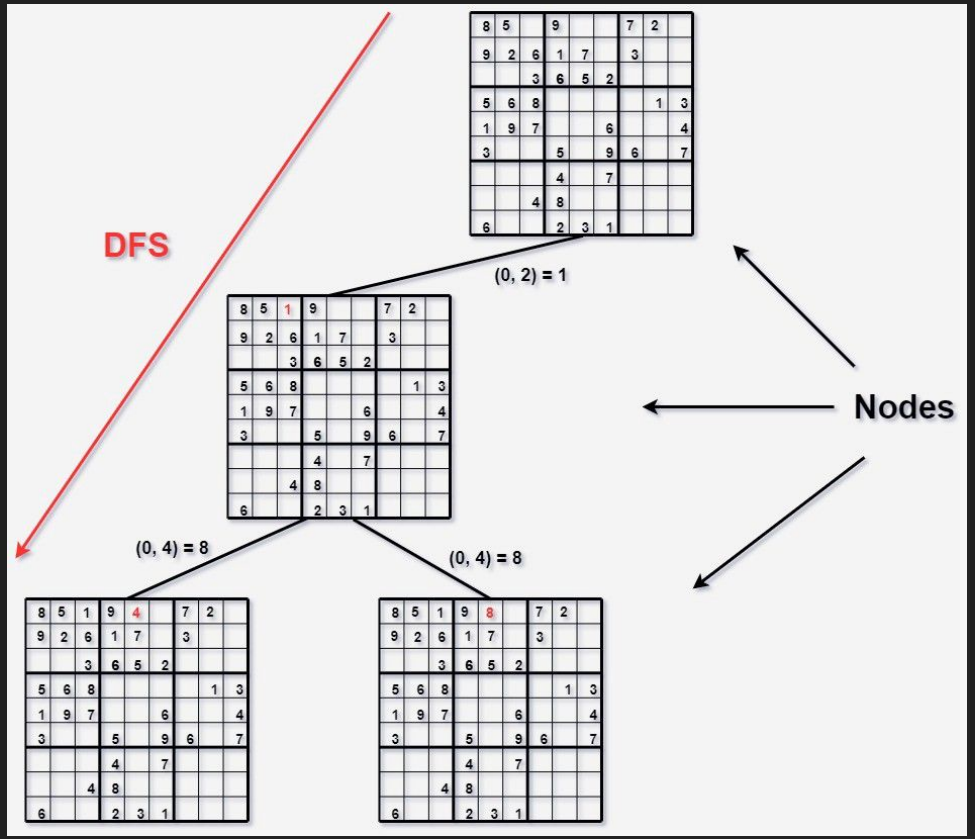


Application au problème

- Choix à faire à chaque case réduisant les possibilités pour les autres

  → si aucune possibilité pour une case, changer la case précédente

- récursivité → choix d'implémentation en OCaml



Structures de données

```
(* le type case_contrainte permet de construire la file de priorite;
le type case_grille est utilise pour modeliser une case*)
type case_contrainte = {mutable nb_contraintes : int; l_ctr : int; c_ctr : int};;
type case_grille = {valeur : int; ligne : int; col : int};;
```

Vérification des cases libres

```
let libre grille ligne col valeur =
    let taille = Array.length grille in
    let cle = int_of_float(sqrt (float_of_int(taille))) in
    let retour = ref true in

    (*Verification Ligne / Colonne *)
        for i = 0 to (taille-1) do
            if ( (grille.(i).(col) = valeur) || (grille.(ligne).(i) = valeur) ) then retour := false;
        done;

    let min_col_boite = (col/cle)*cle in
    let min_ligne_boite = (ligne/cle)*cle in

        for ligne = min_ligne_boite to (min_ligne_boite+cle-1) do
            for col = min_col_boite to (min_col_boite+cle-1) do
                if ( grille.(ligne).(col) = valeur ) then retour := false;
            done;
        done;
    !retour;;
```

//complexité = O(n)

Construction de la file de priorité

```
let priorite grille =
    let taille = Array.length grille in
    let newfile = ref[] in
    for ligne = 0 to (taille-1) do
        for col = 0 to (taille-1) do
            let case = {nb_contraintes = 0; l_ctr = ligne; c_ctr = col} in
            for valeur = 1 to taille do
                if not (libre grille ligne col valeur) then case.nb_contraintes<- case.nb_contraintes + 1;
            done;
            if (grille.(ligne).(col) = 0) then newfile := (inserer !newfile case)
        done;
    done;

    !newfile@[{nb_contraintes = 0; l_ctr = 0; c_ctr = 0}];;
```

//complexité = $O(n^3)$

Résolution du problème - Backtracking

```
let backtracking grille =
    (*declaration / initialisations*)
    let taille = Array.length grille in
    let fileprio = ref (priorite grille) in
    let pile_fils = ref [{valeur = 1; ligne = (List.hd !fileprio).l_ctr; col = (List.hd !fileprio).c_ctr}] in


    let rec backtracking_inside gr pile file_cases depart =
    match !pile with
    |[] -> false
    |s :: d ->
        let ligne = s.ligne in          (*ligne de la case actuelle*)
        let col = s.col in          (*colonne de la case actuelle*)
        let v   = s.valeur in   (*valeur de la case actuelle*)

        match pleine gr with
        |true -> afficher_matrice gr; true
```

Résolution du problème - Backtracking

```
|false ->
            if ((libre gr ligne col val) && (val <= taille)) then begin
                (*si la valeur val est libre et acceptable (inferieure a la taille)
                pour la case de la grille gr d'indice ligne col,
                alors on attribue cette valeur a cette case
                et on etudie la prochaine case dans la file de priorite en commencant par 1*)
                gr.(ligne).(col)<- val;
                let prochain_fils = {
                    valeur = 1;
                    ligne = (List.nth !file_cases (depart + 1)).l_ctr;
                    col = (List.nth !file_cases (depart+1)).c_ctr} in
                backtracking_inside gr (ref (prochain_fils :: d)) file_cases (depart + 1)
            end
```

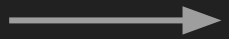
Résolution du problème - Backtracking

```
        end else begin
            (* plusieurs cas se presentent : si val est strictement inferieure a la taille, alors la valeur testee n'est
            pas disponible pour cette case : on teste la valeur suivante*)
            if (val < taille) then
                backtracking_inside gr (ref ({valeur = val+1; ligne = ligne; col = col} :: d)) file_cases depart
            else begin
                (*
sinon, cela signifie que l'on a atteint une feuille de notre arbre : on reassigne donc 0 a la case actuelle et on retourne
                a la case precedente en essayant la valeur superieure*)
                gr.(ligne).(col)<- 0;
                let ligne_anterieure = (List.nth !file_cases (depart-1)).l_ctr in
                let colonne_anterieure = (List.nth !file_cases (depart-1)).c_ctr in
                let valeur_anterieure_maj = gr.(ligne_anterieure).(colonne_anterieure)+1 in
                let prochain_fils = {valeur = valeur_anterieure_maj; ligne = ligne_anterieure; col = colonne_anterieure} in
                backtracking_inside gr (ref (prochain_fils :: d)) file_cases (depart-1)
            end
        end in
backtracking_inside grille pile_fils fileprio 0;;
```

//complexité = $O(n^{n^2})$

Solution :

9			1					5
		5		9		2		1
8				4				0
				8				
			7					
				2	6			9
2			3					6
			2			9		
		1	9		4	5	7	



9	4	3	1	7	2	6	8	5
7	6	5	8	9	3	2	4	1
8	1	2	6	4	5	7	9	3
4	3	6	5	8	9	1	2	7
5	2	9	7	3	1	4	6	8
1	7	8	4	2	6	3	5	9
2	9	4	3	5	7	8	1	6
6	5	7	2	1	8	9	3	4
3	8	1	9	6	4	5	7	2

Conclusion et perspectives (optimisation)

- exhaustivité des solutions

  → parcours entier de l'arbre (complexité)

- différentes tailles de grilles

  → adapter les règles (grille de n^2 cases, n carrés, chiffres de 1 à n)

- remplissage automatique

  → 20% de la grille

  → plus efficace (gain de temps)

MERCI de votre attention

Des questions ?