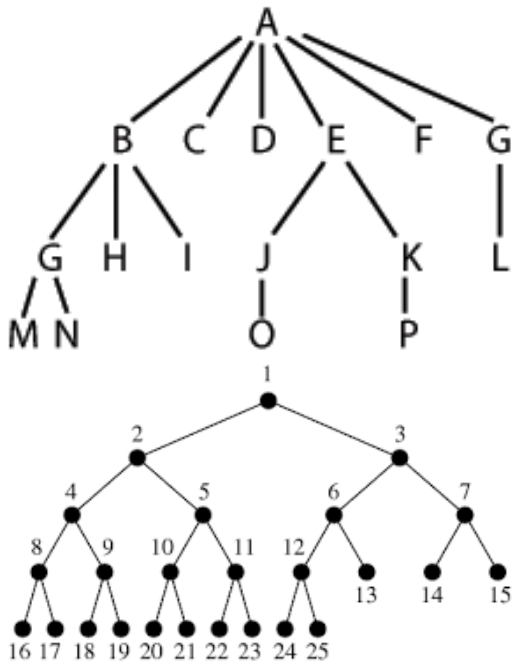
Deuxième cours

Récursivité multiple

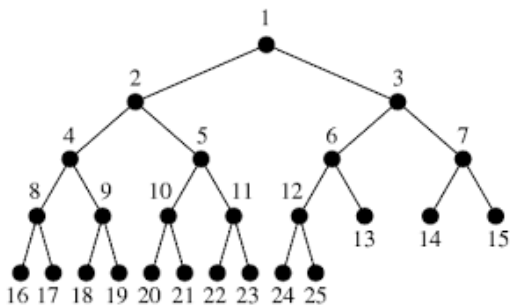
Les arbres

- Structure : Nœud = { valeur, enfants : Set<Nœud> }
- Feuille : Nœud sans enfants.
- arbre *binaire*, arbre *n*-aire.
  = au plus *n* enfants
- arbre *n*-aire *complet* :
  = 0 ou *n* enfants.



Exemple de problèmes sur un arbre :

- Comment calculer la somme des valeurs dans l'arbre ?
- calculer la plus petite valeur d'un arbre.
- vérifier la présence d'une valeur avec une certaine propriété dans l'arbre.



Solution itérative

- Crée une file de nœuds à traiter, au début que la racine.
- Tant que la file n'est pas vide, pour le premier nœud :
  - ajouter ses enfants à la file
  - traiter le nœud
  - le retirer de la file

```
public static int sumAllIter(Node node) {
        int sum = 0;
    LinkedList<Node> queue = new LinkedList<Node>();
    queue.add(node);
        while(!queue.isEmpty()) {
                Node head = queue.pop();
                sum += head.getValue();
                for(Node child : head.getChildren()) {
                        queue.add(child);
                }
        }
        return sum;
}
```

Solution récursive

Stratégie :

- On écrit la fonction en admettant qu'elle fonctionne déjà sur les sous-arbres.

```
int sumAll(Node node){
  int sum = node.getValue();
  for(Node son: node.getChildren()){
    sum += sumAll(son);
    }
  return sum;
}
```

- Où est la condition d'arrêt ?
- Où est la file ?
- ordre de parcours ?

Parcours d'arbre

En profondeur

Pour chaque nœud, on parcourt les enfants avant la fratrie.

En largeur

Pour chaque nœud, on parcourt la fratrie avant la génération suivante.

Parcours en profondeur

Depth First Search (DFS)

```
void depth_first_search(Node node){
  act_on(node);
  for(Node son: node.children){
    depth_first_search(son);
    }
}
```

- Même parcours que précédemment.
- Utilise la pile d'appels comme file d'attente.

Parcours en largeur

Breadth First Search (BFS)

```
void breadth_first_search(Node node){
  LinkedList<Node> queue =  new<LinkedList>;
  queue.add(node);
  while(queue){
    Node current = queue.pop();
    act_on(current);
    for((Node son: node.children)
      queue.add(son);
  }
}
```

Parcours en largeur

Breadth First Search (BFS)

```
void breadth_first_search(Node node){
  LinkedList<Node> queue =  new<LinkedList>;
  queue.add(node);
  while(queue){
    Node current = queue.pop();
    act_on(current);
    for((Node son: node.children)
      queue.add(son);
  }
}
```

- Que se passe-t-il si on remplace le add par un push ?

(différence entre *pile* et *file*)

Parcours d'arbre

En profondeur

Pour chaque noeud, on parcourt les enfants avant la fratrie.

- Appels suivants dans une **pile**.

En largeur

Pour chaque noeud, on parcourt la fratrie avant la génération suivante.

- Appels suivants dans une **file**.

Parcours d'arbre

En profondeur

Pour chaque noeud, on parcourt les enfants avant la fratrie.

- Appels suivants dans une **pile**.
- longueur de la pile = profondeur de l'arbre.

En largeur

Pour chaque noeud, on parcourt la fratrie avant la génération suivante.

- Appels suivants dans une **file**.
- Longueur de la file = largeur de l'arbre.

Ordre de traitement

Parfois utile de déplacer l'appel de `act_on`.

prefixe : on traite le parent avant les enfants (en descendant).

postfixe : on traite le parent après les enfants (en remontant).

infixe : on traite le parent au milieu (au cours du demi tour).

- calcul de strategie : postfixe.
- exploration : préfixe.

Solveur de sudoku

Comment résoudre un sudoku ?

Méthode explicite

Essayer de coder les règles que l'on appliquerai *à la main*

- un seul élément possible dans une case
- une seule case possible pour un élément
- principe d'inclusion – exclusion.

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

Solveur de sudoku

Comment résoudre un sudoku ?

Méthode explicite

Essayer de coder les règles que l'on appliquerai *à la main*

- un seul élément possible dans une case
- une seule case possible pour un élément
- principe d'inclusion – exclusion.
- pas toujours suffisant.

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

Méthode explicite

À quoi ressemblerait le code ?

```
boolean fill(Sudoku sudoku) throws IllegalStateException{
    boolean progress = true;
    while(progress && !game_over(sudoku)){
      progress = search_move_exclusion(sudoku);
      progress = progress || search_move_inclusion(sudoku);
    }
    if (!game_over(sudoku)){
      throw(new IllegalStateException("Failed"));
    }
    else
      return is_valid(sudoku);
}
```

Solveur de sudoku

Comment résoudre un sudoku ?

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

Retour sur trace – backtrack

Tout essayer ! (recherche exhaustive)

Pour chaque case :

1. placer la prochaine valeur disponible
2. continuer sur le reste
3. si aucune solution trouvée, retourner en 1.

Retour sur trace – backtrack

À quoi ressemblerait le code ?

```
boolean solve(Sudoku sudoku){
    if(game_over(sudoku))
     return is_valid(sudoku);
   else {
     Square choice = first_empty_square(sudoku);
     for(int i = 1; i <= 9; i++){
        sudoku.set(choice, i);
        if(solve(sudoku))
          return true;
     }
     sudoku.unset(choice);
     return false;
   }
}
```

Simulation de l'algorithme

Au tableau

Compréhension

- État du sudoku à la fin de l'exécution ?
- Profondeur de la pile ?

Simulation de l'algorithme

Au tableau

Compréhension (++)

- État du sudoku à la fin de l'exécution ?
- Profondeur de la pile ?
- Estimation du nombre d'appels récursifs ?
- Ordre de parcours ?

Simulation de l'algorithme

Au tableau

Compréhension (++)

- État du sudoku à la fin de l'exécution ?
- Profondeur de la pile ?
- Estimation du nombre d'appels récursifs ?
- Ordre de parcours ?

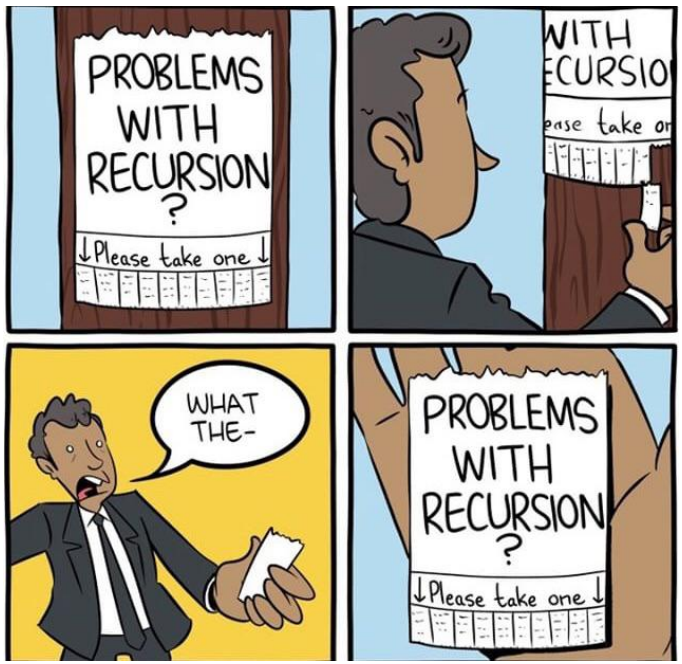
Adaptabilité

- Utilisable pour compter/lister les solutions
- Optimisation : éviter des explorations inutiles

Synthèse

À retenir

- Appels récursifs pour traiter tous les nœuds d'un arbre.
- Ordres de parcours, subtilités des parcours.
- Méthode de retour sur trace, backtrack.



`https://www.smbc-comics.com/comic/recursion`