Structure de données: Pile et File

I) La structure de pile

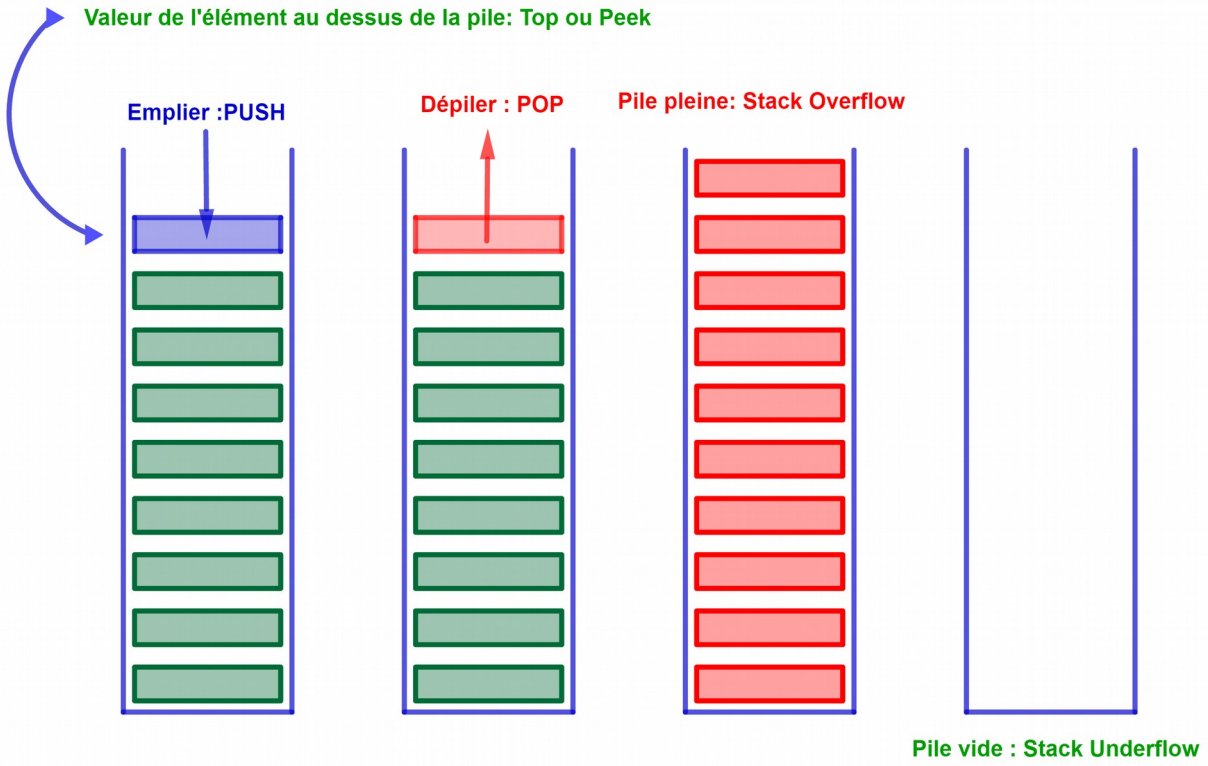
La documentation provient du site **GeekforGeeks** (data structure: **stack**)

https://www.geeksforgeeks.org/stack-data-structure/

La pile est une structure de données linéaire qui obéit à des règles particulières en termes d'opérations. En anglais on résume ce processus par la méthode:

LIFO : Last In First Out (dernier entré, premier sorti)

Pour construire la pile, on utilise une structure de tableau. Ce dernier admet une taille maximale. On peut implémenter les méthodes suivantes



1. **empiler**: On ajoute un élément à la fin du tableau si ce dernier n'est pas rempli, sinon on renvoie un message de "pile pleine" (**stack overflow**).

2. **dépiler**: On retire un élément en fin de tableau si ce dernier n'est pas vide, sinon on renvoie un message de "pile vide" (**stack underflow**).

3. **estVide**: On teste si la pile est vide.

4. **dessus**: On renvoie la valeur en haut de pile (**peek/top**).

II) La structure de file

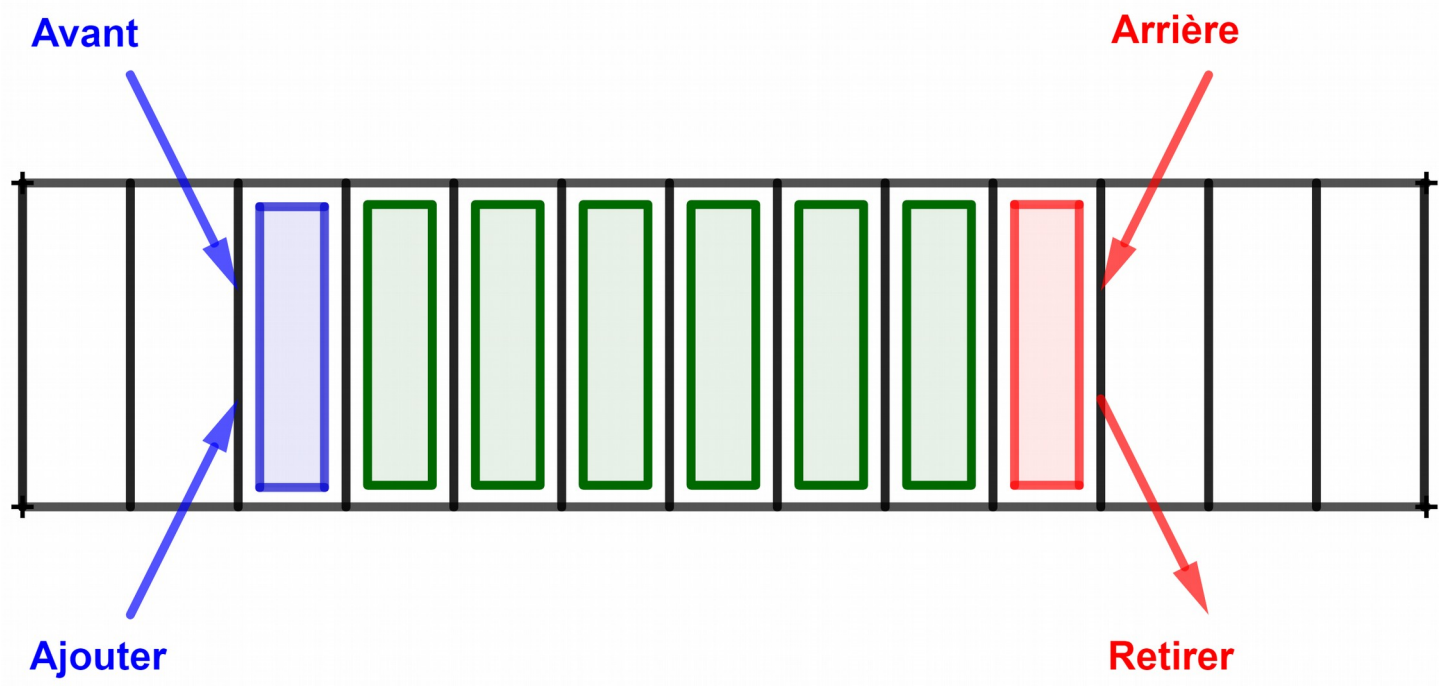
La documentation provient du site **GeekforGeeks** (data structure: **queue**)

https://www.geeksforgeeks.org/queue-data-structure/

La file est aussi une structure de données linéaire. Elle reste assez proche d'une pile (notion de taille et de file vide) mais diffère de par l'ajout et le retrait de ses éléments. En anglais on résume ce processus par la méthode:

FIFO: First In First Out (dernier entré, premier sorti)

Pour construire la file, on utilise une structure de tableau. Ce dernier admet une taille maximale. On peut implémenter les méthodes suivantes



1. **ajouter** : On ajoute un élément au début du tableau si ce dernier n'est pas plein, sinon on renvoie un message de "file pleine" (**queue overflow**)

2. **retirer** : On retire un élément en fin de tableau si ce dernier n'est pas vide, sinon on renvoie un message de "file vide" (**queue underflow**)

3. **estVide** : On teste si la file est vide.

4. **avant** : On renvoie la valeur en début de file (**front**).

5. **arrière** : On renvoie la valeur en file de file (**rear**).

III) Les applications

On donne ici une liste d'exemples et d'applications des structures de pile et file.

1) Vérification du parenthésage :

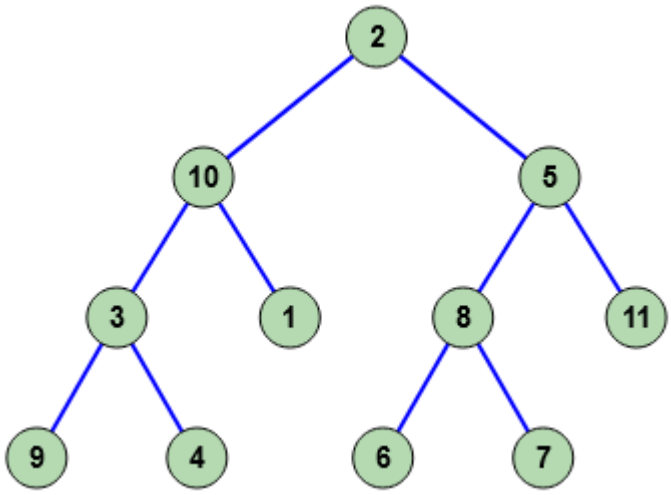
On construit une pile qui empile les parenthèses ouvrantes. On dépile la parenthèse si on rencontre une parenthèse fermante. On stop le processus si un couple de parenthèse ne se correspondent pas.

2) Bouton reculer/avancer :

Une implémentation des boutons **reculer/avancer** dans les navigateurs repose sur l'implémentation de 2 piles. On peut aussi implémenter les boutons (**Do/Undo**) dans un éditeur de texte.

3) Parcours d'un arbre :

On peut utiliser une méthode diviser pour régner (récursivité). On retiendra 3 formats d'implémentations.



- **Ordre:** Gauche Racine Droit : 9 – 3 – 4 – 10 – 1 – 2 – 6 – 8 – 7 – 5 - 11
- **Préordre:** Racine Gauche Droit : 2 – 10 – 3 – 9 – 4 – 1 – 5 – 8 – 6 – 7 - 11
- **Postordre:** Gauche Droit Racine : 9 – 4 – 3 – 1 – 10 – 6 – 7 – 8 – 11 – 5 - 2

Cet arbre a été réaliser sur le site yEdlive https://www.yworks.com/yed-live/

4) Aire sous un histogramme :

C'est un algorithme de recherche qui consiste à déterminer un rectangle d'aire maximale sous un histogramme. On l'implémente avec une méthode diviser pour régner. On parcourt l'histogramme en recherchant une valeur minimale (gauche/droite).

5) Un plateau d'échec (Bactracking) :

Sur un échiquier on place des pièces de même type: Cavalier, Dame, Tour. Il faut remplir l'échiquier avec un maximum des pièces sans prises mutuelles. On utilise le principe du backtracking. On construit une pile des positions qui teste si la prise est possible ou pas. On explore les différents choix possibles (très long) et on retient les meilleurs cas.