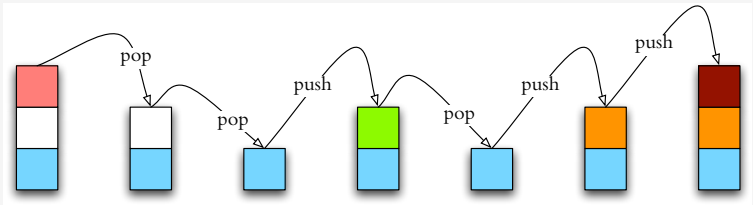
IFT1025 été 2010

Miklós Csűrös

28 juillet 2010

14 Structures de données II

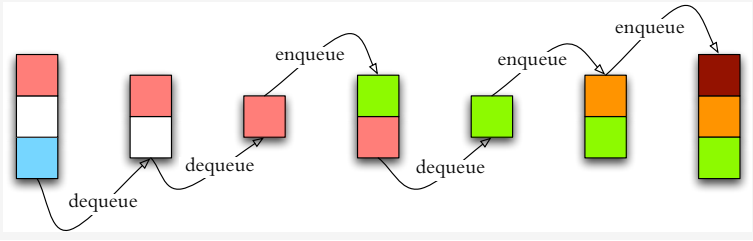
14.1 Structures de donées élémentaires



Pile : objets empilés un sur l'autre, on ne peut accéder qu'à l'élément supérieur.
opérations : «empiler» (push) et «dépiler» (pop), teste «vide ?» (isEmpty).

```
public class Stack extends LinkedList // implantation par liste chaînée
{
    public Stack(){super();} // pile vide

    public boolean isEmpty()
    {
        return (premier == null);
    }
    public void push(Object o)
    {
     insertFirst(o);
    }
    public Object pop()
    {
        if (isEmpty()) throw new EmptyStackException(); // erreur: RuntimeException
        Object retval = premier;
        deleteFirst();
        return retval;
    }
}
```



Queue ou file d'attente : objets rangés un après l'autre, on peut enfiler à la fin ou défiler au début.
opérations : «enfiler» (enqueue) et «défiler» (dequeue), teste «vide ?» (isEmpty).

Implantation par liste chaînée : maintenir une référence au dernier élément sur la liste. Enfiler : insertion à la fin ; défiler : délétion à la tête.

14.2 Arbres binaires de recherche

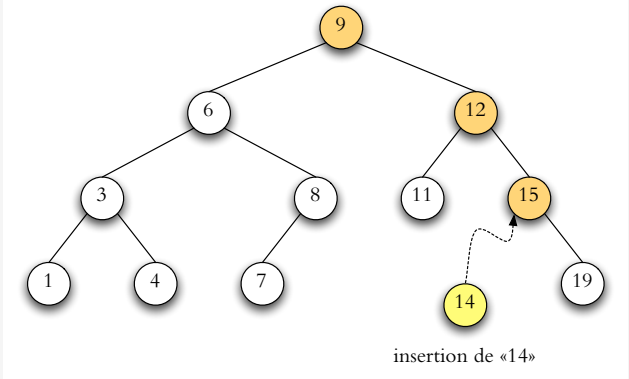
Arbre binaire de recherche : implantation d'un dictionnaire (ou tableau de symbôles)
Recherche binaire dans un tableau
Mimiquer la recherche à l'aide de références «gauche» et «droite» associées avec les nœuds.

```
public class ABR // arbre binaire de recherche
{
    public ABR(){ racine = null;} // arbre vide
    private static class Noeud
    {
        int clef;
        Object valeur;
        Noeud gauche;
        Noeud droit;
    }

    private Noeud racine;

    private Noeud recherche(int clef)
    {
        Noeud N = racine;
        while (N!=null)
        {
            if (N.clef == clef)
                break;
            if (clef < N.clef)
                N = N.gauche;
            else
                N = N.droit;
        }
        return N;
    }

}
```



insertion de «14»

insertion :comme **recherche**(), on trouve la place pour v (enfant gauche ou droit manquant).

Efficacité de recherche et mise à jour : temps de calcul dépend de la hauteur de l'arbre
$\Rightarrow$ ABR efficaces avec maintenance d'équilibre pour avoir hauteur $\sim \log n$.