

- Arbres de segments
- Arbres de sélection
- Listes à saut
- Compléments de Java
- Dictionnaires
- Automates

Problème : Chercher, dans un ensemble d'intervalles de la droite réelle, les intervalles contenant un réel donné.

Applications :

- En géométrie algorithmique.
- En gestion de documents ou de logiciels : chaque version est valable pendant une durée fixe.

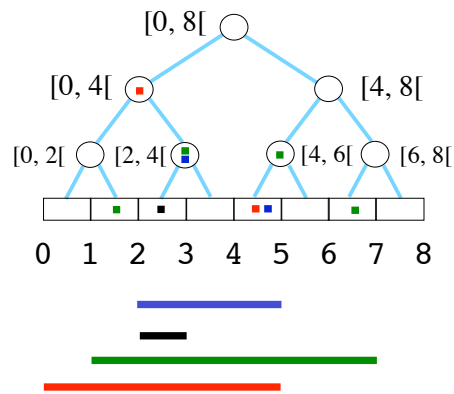
Problème : Chercher, dans un ensemble de n intervalles de la droite réelle, les intervalles contenant un réel donné.

Solutions :

- naïve, en temps $O(n)$: on parcourt la liste des intervalles,
- par un **arbre de segments**, en temps $O(\log n)$ avec un prétraitement en $O(n \log n)$

- Les intervalles sont de la forme $[g, d[$ (fermés à gauche, ouverts à droite).
- Ils peuvent se chevaucher, et partager des extrémités.
- Les extrémités sont triées, et identifiées aux entiers de 0 à m (avec $m \leq 2n$).

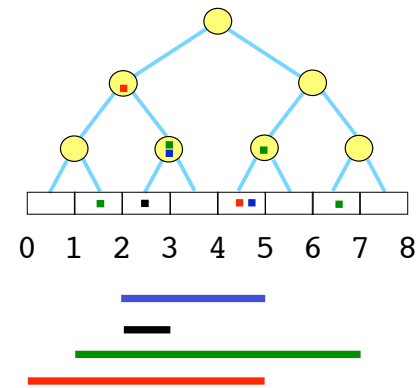
Exemple



Chaque intervalle est codé par un certain nombre de **jetons**.

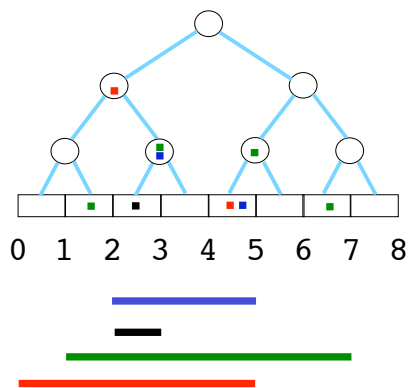
$$\begin{aligned}
 [0, 5] &= [0, 4[\cup [4, 5[\\
 [1, 7] &= [1, 2[\cup [2, 4[\\
 &\quad \cup [4, 6[\cup [6, 7[
 \end{aligned}$$

Prétraitement d'un intervalle



- (1) **Découper** en intervalles élémentaires.
- (2) **Remonter** un jeton si ses deux fils ont un jeton de même couleur.

Nombre d'intervalles contenant un point



Le nombre d'intervalles contenant un point x est le nombre de jetons sur le chemin vers la feuille contenant x

Exemple : $x = 2.5$
Nombre d'intervalles = 4

Structure d'arbre

```

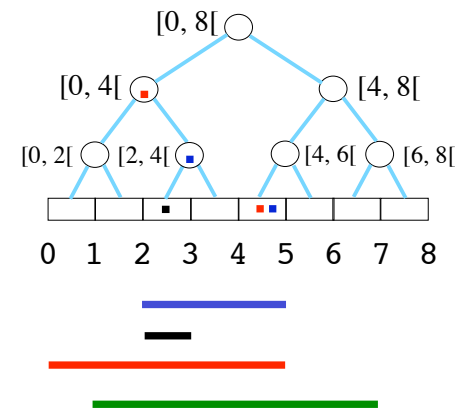
class Arbre
{
    int g, d; // intervalle du noeud
    int n;    // nombre de jetons
    Arbre filsG, filsD;

    Arbre(int gg, int dd,
          Arbre ag, Arbre ad)
    {
        g = gg; d = dd;
        filsG = ag; filsD = ad;
        n = 0;
    }
}
    
```

Compter les intervalles

```
static int compter(Arbre a, double x)
{
    if (a.filsG == null
        && a.filsD == null)
        return a.n ;
    int m = (a.g + a.d)/2;
    if (x < m)
        return a.n + compter(a.filsG, x);
    return a.n + compter(a.filsD, x);
}
```

Insertion d'un intervalle



Insérer un intervalle aux extrémités connues

```
static void inserer(Arbre a, int g, int d)
{
    if (a.g == g && a.d == d) a.n++;
    else {
        int m = (a.g + a.d)/2;
        if (d <= m)
            inserer(a.filsG, g, d);
        else if (g >= m)
            inserer(a.filsD, g, d);
        else { // g < m < d
            inserer(a.filsG, g, m);
            inserer(a.filsD, m, d);
        }
    }
}
```

Plan

- Arbres de segments
- Arbres de sélection
- Listes à saut
- Compléments de Java
- Dictionnaires
- Automates

Fusion de k listes ordonnées

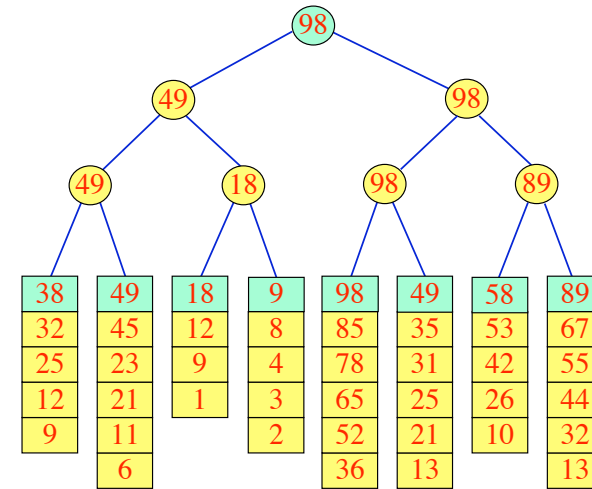
Objectif : fusionner les 8 listes ordonnées ci-dessous en une seule liste ordonnée.

38	49	18	9	98	49	58	89
32	45	12	8	85	35	53	67
25	23	9	4	78	31	42	55
12	21	1	3	65	25	26	44
9	11		2	52	21	10	32
	6			36	13		13

Amphi 9

13

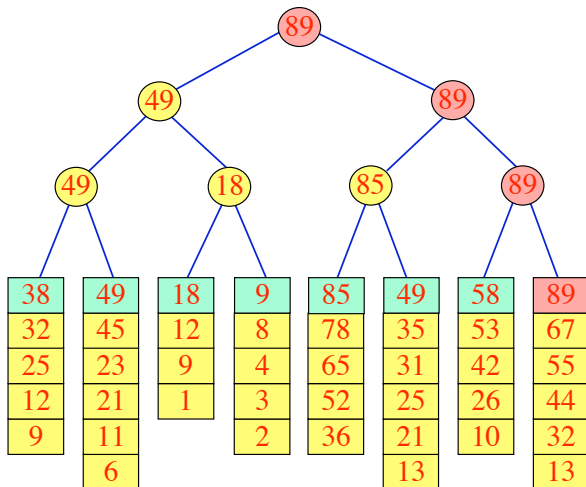
Fusion de k listes ordonnées



Amphi 9

14

Mise à jour de l'arbre de sélection



Amphi 9

15

Performances des arbres de sélection

Si k est le nombre de listes, la création de l'arbre de sélection est en $O(k)$. La hauteur de l'arbre est $O(\log k)$. Le temps nécessaire pour restructurer l'arbre est donc aussi $O(\log k)$.

La fusion de n éléments se fait en $O(n \log k)$.

La fusion de k listes se fait donc en $O(k + n \log k)$.

L'algorithme naïf est en $O(kn)$.

Amphi 9

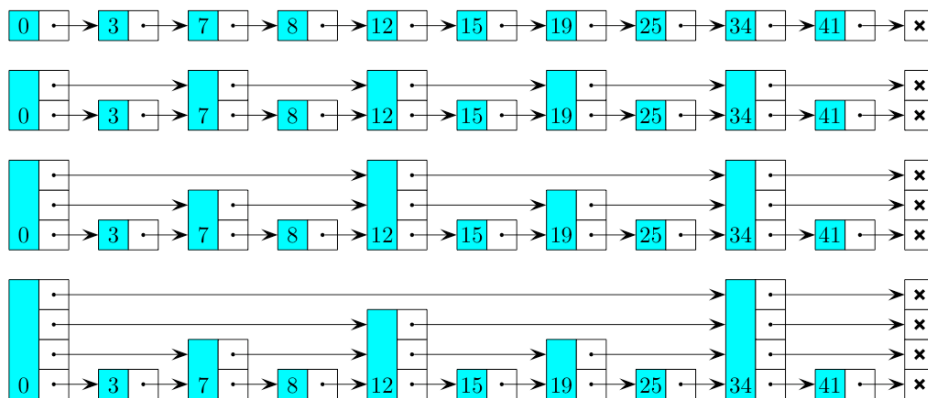
16

- Arbres de segments
- Arbres de sélection
- **Listes à saut**
- Compléments de Java
- Dictionnaires
- Arbres des suffixes
- Automates



William Pugh

Skip Lists: A Probabilistic Alternative to Balanced Trees.
Commun. ACM 33 (1990), 668-676



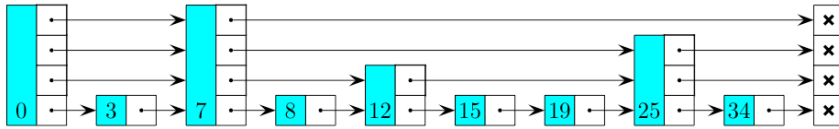
Accélère la recherche dans une liste **ordonnée**

Dans une **liste à niveaux**:

- le niveau 0 contient tous les éléments,
- le niveau 1 contient un élément sur **2**,
- le niveau 2 contient un élément sur **4**,
- le niveau 3 contient un élément sur **8**, etc.

La **recherche** est en $O(\log n)$, mais l'**insertion** et la **suppression** sont en $O(n)$.

Liste à saut



Même principe que les listes à niveau, mais les niveaux sont choisis de façon aléatoire de façon à ce que $P(\text{niveau} = n) = p^{-(n+1)}$ (où $0 < p < 1$)
Par exemple, si $p = 1/2$, le niveau sera 2 avec une probabilité de 2^{-3} .

Niveau aléatoire

```
static final float PROBA = 0.5f;
static final int NMAX= 20;
// Valable jusqu'à 2^20 nœuds

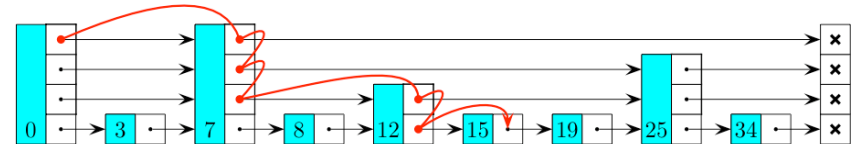
static int niveauAleatoire() {
    int niveau = 0;
    while (niveau < NMAX&&
           Math.random() < PROBA)
        niveau++;
    return niveau;
}
```

Les listes à saut en Java

```
class ListeASaut
{
    int contenu;
    ListeASaut[] tableau;

    ListeASaut(int niveau, int contenu)
    {
        this.contenu = contenu;
        tableau = new ListeASaut[niveau + 1];
    }
}
```

Recherche dans une liste à saut



Recherche de 19

```
static boolean estDans(int k, ListeASaut a)
{
    return (a != null) &&
           estDans(a.tableau.length-1, k, a);
}
```

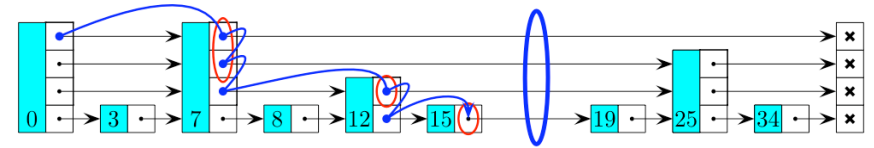
Recherche dans une liste à saut (2)

```
static boolean estDans(int n, int k,
                      ListeASaut a) {
    // n désigne le niveau
    if (a == null || a.contenu > k || n < 0)
        return false;
    if (a.contenu == k)
        return true;
    if (a.tableau[n] == null ||
        a.tableau[n].contenu > k)
        return estDans(n - 1, k, a);
    return estDans(n, k, a.tableau[n]);
}
```

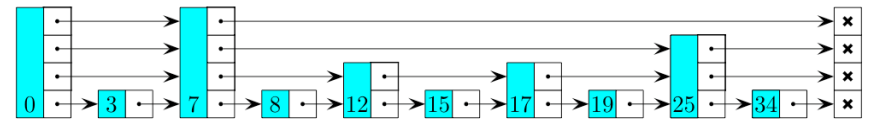
Amphi 9

25

Insertion dans une liste à saut



Insertion de 17 : on calcule le "front avant", puis on insère avec un niveau aléatoire (ici 1).



Amphi 9

26

Calcul du front avant

```
static ListeASaut[] FrontAvant(int k,
                               ListeASaut a)
{
    ListeASaut[] t = new ListeASaut[NMAX+1];
    for (int i = NMAX; i >= 0; i--) {
        while (a.tableau[i] != null &&
              a.tableau[i].contenu < k)
            a = a.tableau[i];
        t[i] = a;
    }
    return t;
}
```

Amphi 9

27

Insertion dans une liste à saut non vide

```
static void inserer(int k, ListeASaut a)
{ // suppose a non vide
    ListeASaut[] t = FrontAvant(k, a);
    int niveau = niveauAleatoire();
    ListeASaut b = new ListeASaut(niveau, k);
    for (int i = 0; i <= niveau; i++)
    {
        b.tableau[i] = t[i].tableau[i];
        t[i].tableau[i] = b;
    }
}
```

Amphi 9

28

Suppression dans une liste à saut

```
static void supprimer(int k, ListeASaut a)
{
    ListeASaut[] t = FrontAvant(k, a);
    ListeASaut b = t[0].tableau[0];
    if (b.contenu == k)
        for (int i = 0; i <= NMAX; i++)
            if (t[i].tableau[i] == b)
                t[i].tableau[i] = b.tableau[i];
}
```

Complexité des opérations

On montre que l'espérance du temps pour les opérations de **recherche**, **insertion** et **suppression** est en $O(\log n)$.

Les listes à saut sont un type de donnée **probabiliste**. L'utilisation d'algorithmes probabilistes est fréquente en informatique : le plus célèbre est le test de primalité de Miller-Rabin (1976).

Plan

- Arbres de sélection
- Listes à saut
- Arbres de segments
- **Compléments de Java**
- Dictionnaires
- Automates

this

Le mot clé **this** désigne l'objet courant. Il n'a jamais la valeur **null**, et sa valeur ne peut être modifiée. Il est fréquemment utilisé dans les constructeurs.

```
class Personne {
    int age;
    float poids;
    float taille;

    Personne(int age, float taille){
        this.age = age;
        this.taille = taille;
    }
    // ... à suivre
```

```
Toto(int age, float poids, float taille){
    this.taille, age);
    this.poids = poids;
}
} // Version correcte.
```

```
Toto(int age, float poids, float taille){
    this.poids = poids;
    this.taille, age);
} // error ! call to this must be
} // first statement in constructor
```

- Arbres de sélection
- Listes à saut
- Arbres de segments
- Compléments de Java
- **Dictionnaires**
- Automates

Comment représenter en machine un ensemble de mots ? {bal, ban, bals, bans, pal, pals, pan, pans}

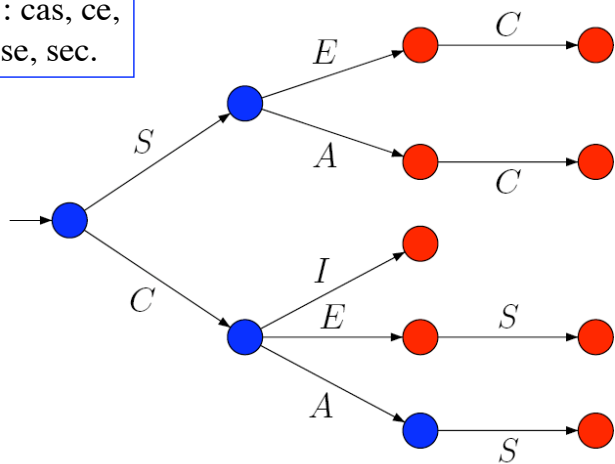
Par tableau (ou par liste)

bal	ban	bals	bans	pal	pals	pan	pans
-----	-----	------	------	-----	------	-----	------

Par arbre digital ("trie" en anglais).

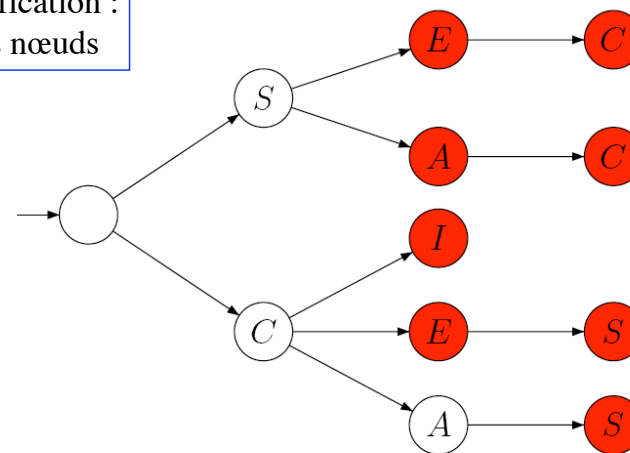
Arbre digital (Trie)

Mots reconnus : cas, ce, ces, ci, sa, sac, se, sec.



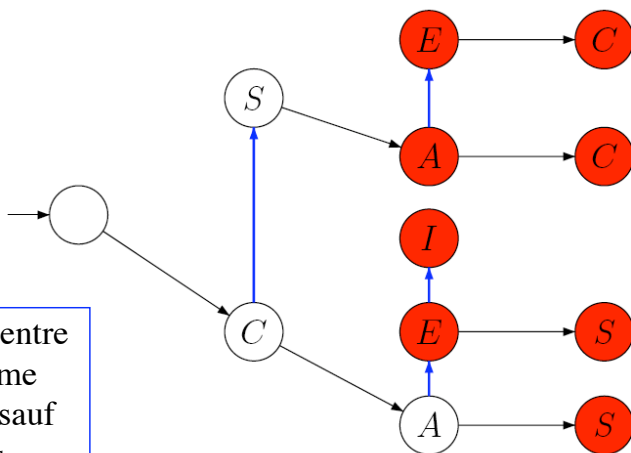
Représentation des arbres digitaux

Première modification : on étiquette les nœuds



Représentation des arbres digitaux

Transformation en arbre binaire.



On trace des liens entre frères et on supprime les liens père-fils, sauf vers le premier fils.

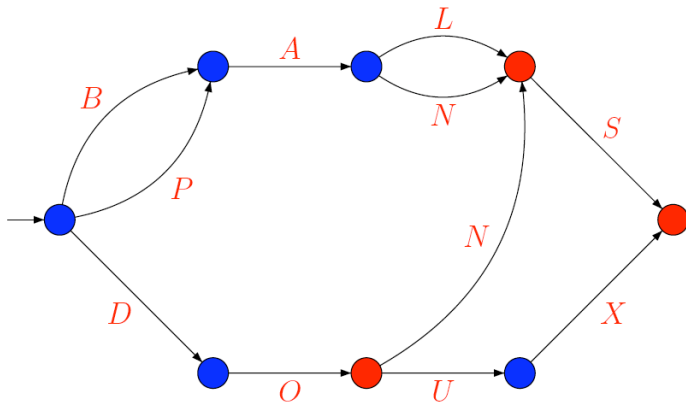
Représentation des arbres digitaux

```
class Digital {
    char lettre;
    boolean estFinal;
    Digital fils, filsSuivant;

    Digital(char lettre) {
        this.lettre = lettre;
    }

    Digital(char lettre, Digital fils, Digital s){
        this.lettre = lettre;
        this.fils = fils;
        this.filsSuivant = s;
    }
}
```

Représentation par automate



Un automate pour l'ensemble de mots {bal, ban, bals, bans, pal, pals, pan, pans, do, don, dons, doux}.

Comparaison

Pour un dictionnaire de Scrabble™ anglais, la représentation par arbre digital nécessite 117 150 nœuds et 780 kilooctets de mémoire. La représentation par automate nécessite seulement 19 853 états et 175 koctets de mémoire, soit un gain de près de 80%.

Plan

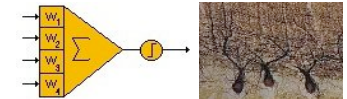
- Arbres de sélection
- Listes à saut
- Arbres de segments
- Compléments de Java
- Dictionnaires
- Automates

La théorie des automates : historique

1949: Shannon (théorie de l'information)



1943 : McCulloch et Pitts (réseaux neuronaux).

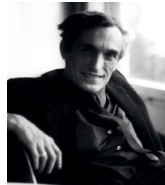


1956 : travaux fondateurs de Kleene et Chomsky



La théorie des automates : historique

1960-1980: développements, notamment en France par l'école de Schützenberger.

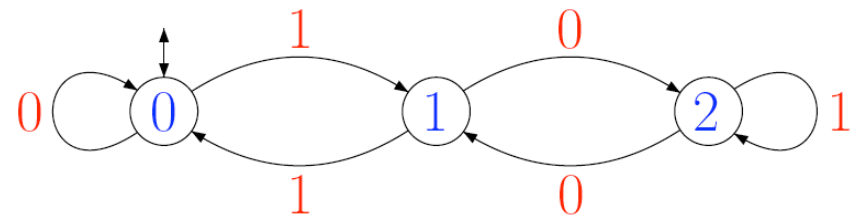


Depuis 1980, nouveaux horizons :

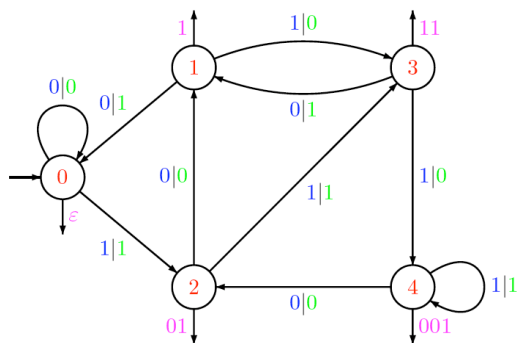
- **Mathématiques** : logique, algèbre, topologie, systèmes dynamiques.
- **Informatique** : compilation, vérification, industries de la langue, circuits booléens, etc.

Un exemple : multiples de 3, en base 2

Le nombre 330 s'écrit 101001010 en base 2.



La multiplication par 5



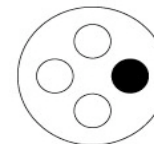
$13 \times 5 = 65$
Entrée : 1011
Sortie : 100001

Cet automate réalise la multiplication par 5 en binaire **inversé**.

Un petit casse-tête (1)

Un joueur a les yeux bandés. Face à lui, un **plateau circulaire** sur lequel sont disposés en carré quatre jetons, blancs d'un côté et noirs de l'autre. La configuration de départ est inconnue du joueur.

Le but du jeu est d'avoir les **quatre jetons** du côté blanc.



Configurations possibles, à une rotation près.

Règles du jeu

Le joueur peut retourner autant de jetons qu'il le souhaite, mais sans les déplacer. A chaque tour, le maître de jeu annonce si la configuration obtenue est gagnante ou pas, puis effectue une rotation du plateau de 0, 1, 2 ou 3 quarts de tours.

Peut-on gagner à coup sûr ? En combien de coups ?

