Les arbres binaires

Arbres de recherche binaire - Applications - Implémentation

D.E ZEGOUR

Ecole Supérieure d'Informatique

ESI

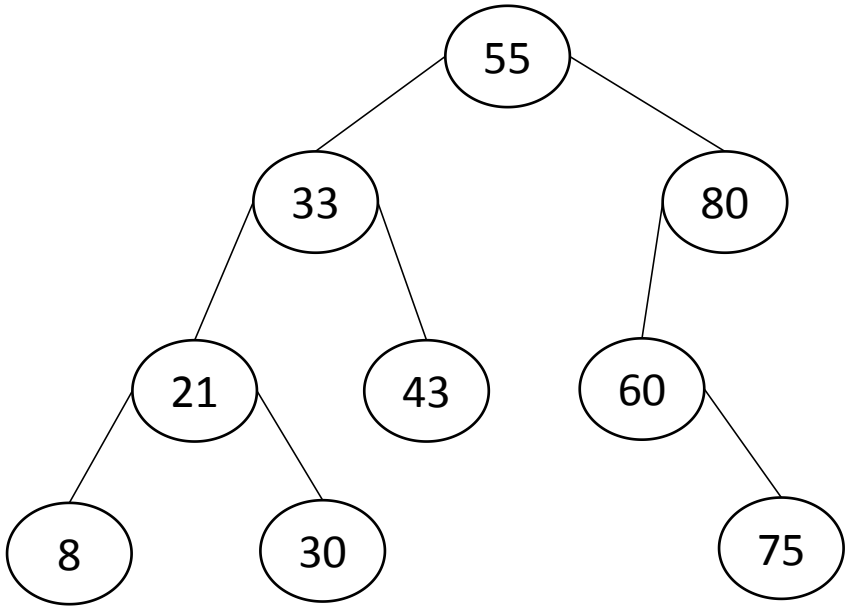
Les arbres binaires

Arbre de recherche binaire

Représente un ensemble ordonné par une relation d'ordre notée < (au sens strict)

- Tous les éléments stockés dans le sous arbre gauche d'un noeud contenant x sont strictement inférieurs à x
- Tous les éléments rangés dans le sous arbre droit d'un noeud contenant x sont strictement supérieurs à x.

Propriété : le parcours en inordre (T1nT2) d'un arbre de recherche binaire donne la liste ordonnée de tous ses éléments.

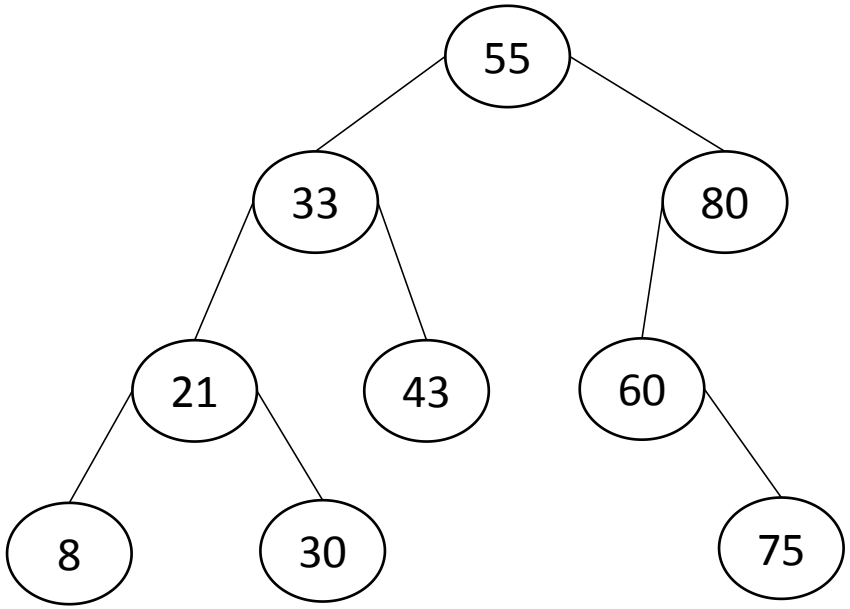


Inordre : 8, 21, 30, 33, 43, 55, 60, 75, 80

Les arbres binaires

Arbre de recherche binaire : Recherche

```
R(A, X)
  SI A = NIL
        R := FAUX
  SINON
        SI X = Info(A)
              R := VRAI
           SINON
              SI X < Info(A)
                    R := R (X, Fg(A))
                 SINON
                    R := R (X, Fd(A))
              FSI
        FSI
FSI
```



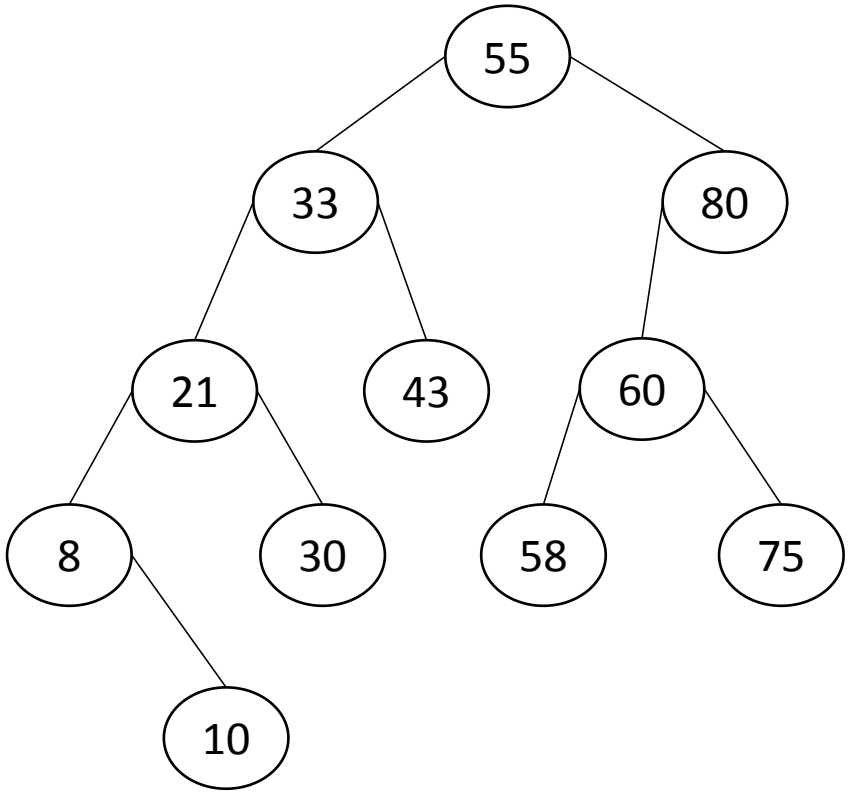
Complexité : O(Log2(n))

Les arbres binaires

Arbre de recherche binaire : Insertion

L'élément inséré est toujours une feuille.

Insertion de 10, puis 58



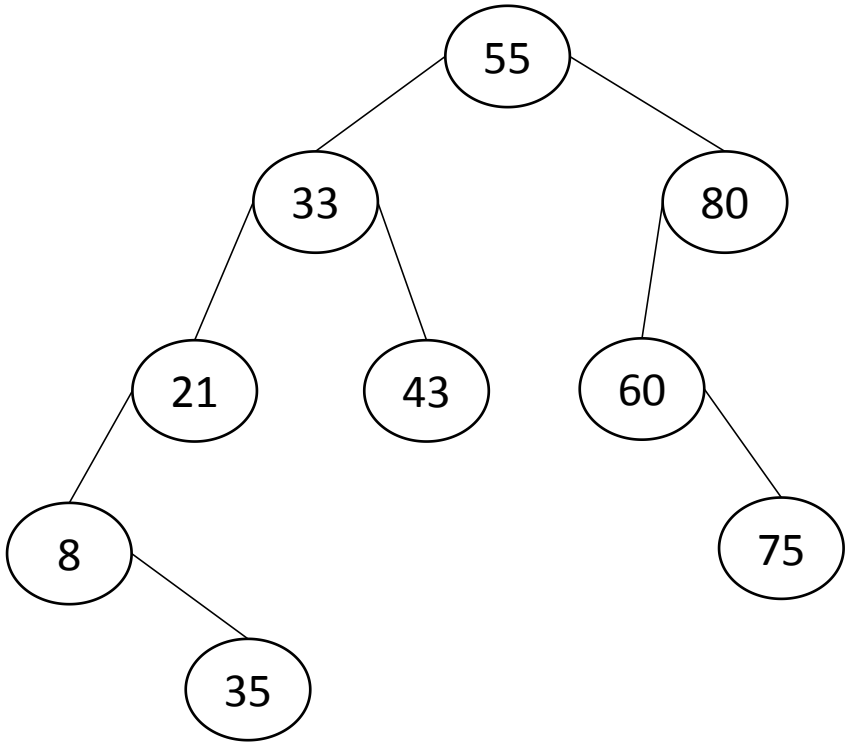
Complexité : O(1)

Les arbres binaires

Arbre de recherche binaire : Suppression

Plusieurs cas à considérer:
soit n le noeud a supprimer

fg(n) = nil et fd(n)= nil	Remplacer n par nil
fg(n) = nil et fd(n) # nil	Remplacer n par fd(n)
fg(n) # nil et fd(n)= nil	Remplacer n par fg(n)
fg(n) # nil et fd(n) # nil	Soit p le successeur inordre de n - Remplacer info(n) par Info(p) - Remplacer p par fd(p)



Complexité : O(Log2(n))

Les arbres binaires

Application 1 : Recherche des doubles

En entrée : une suite de nombres à lire
En sortie : Afficher les doubles

Pour détecter les doubles, il faudrait sauvegarder les nombres dejà lus

Problème : Où les ranger ?

Pseudo algorithme
Soit S la structure de données choisie
Pour chaque nombre n lu:
    Rechercher n
    Si n existe dans S
        Ecrire (n)
    Sinon
        Inserer(n) dans S
    Fsi
Fpour

Les arbres binaires

Application 1 : Recherche des doubles

Solution 1 : Tableau
- non ordonné : recherche séquentielle : O(n) et insertion à la fin : O(1)
- ordonné : recherche dichotomique : O(Log2(n)) et insertion avec décalage O(n)

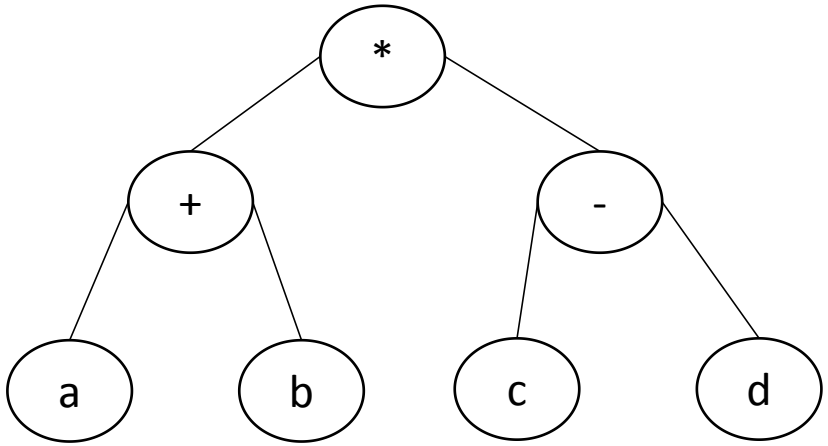
Solution 2 : Listes linéaires chaînées
- non ordonné : recherche séquentielle : O(n) et insertion à la fin : O(1)
- ordonné : recherche séquentielle : O(n) et insertion au bon endroit : O(1)

Solution 3 : Arbre de recherche binaire
- recherche dichotomique : O(Log2(n)) et insertion O(1)

Les arbres binaires

Application 2 : Expressions arithmétiques

Une expression arithmétique peut être représentée par un arbre binaire
Opérateurs : noeuds internes
Opérandes : noeuds externes



(a + b) * (c - d)

Parcours préordre : * + a b - a b (notation polonaise préfixée)

Parcours postordre : a b + c d - * (notation polonaise postfixée)

Parcours inordre : a + b * c - d (expression sans paraenthèse)

Les arbres binaires

Application 2 : Expressions arithmétiques (algorithme)

Soit A un arbre représentant une expression arithmétique.

L'algorithme utilise un parcours postordre

Oper(Op, A, B) une fonction qui effectue l'opération Op entre deux valeurs A et B

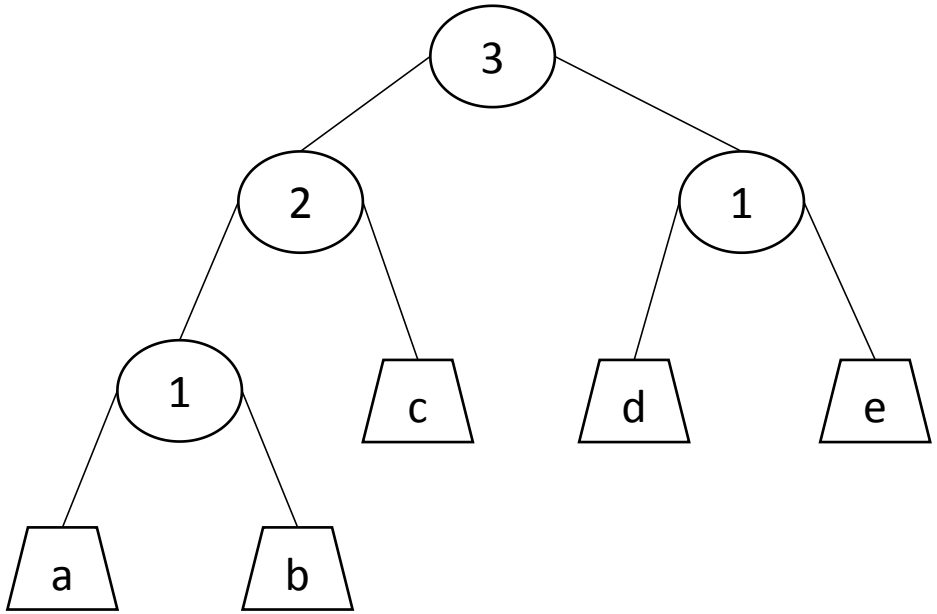
```
Eval(A) :
SI Feuille(A) :
        Eval := Info(A)
   SINON
        Eval := Oper(Info(A), Eval(Fg(A), Eval(Fd(A))
FSI
```

Les arbres binaires

Application 3 : Représentation des listes par des arbres binaires

Une liste peut être représenté par un arbre:
feuilles : éléments de la la liste
Noeuds internes : nombre de feuilles dans le sous arbre gauche

But : Retrouver rapidement le k-ième élément

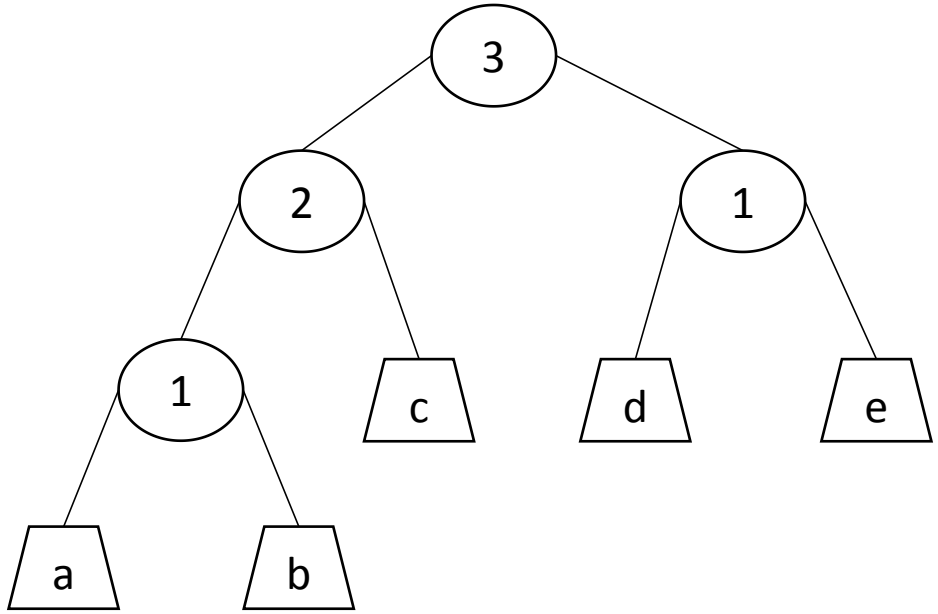


Les arbres binaires

Application 3 : Représentation des listes par des arbres binaires

Algorithme : Recherche du k-ième élément

```
R := K
P := Arbre
TANTQUE NON Feuille(P) :
      SI R <= Compte(P) :
           P := Fg(P)
        SINON
           R := R - Compte(P)
           P := Fd(P)
      FSI
FINTANTQUE
```



Feuille (P) : Vrai si P est une feuille, faux sinon
Compte(P) : Nombre de feuilles dans le sous arbre gauche du noeud P

Les arbres binaires

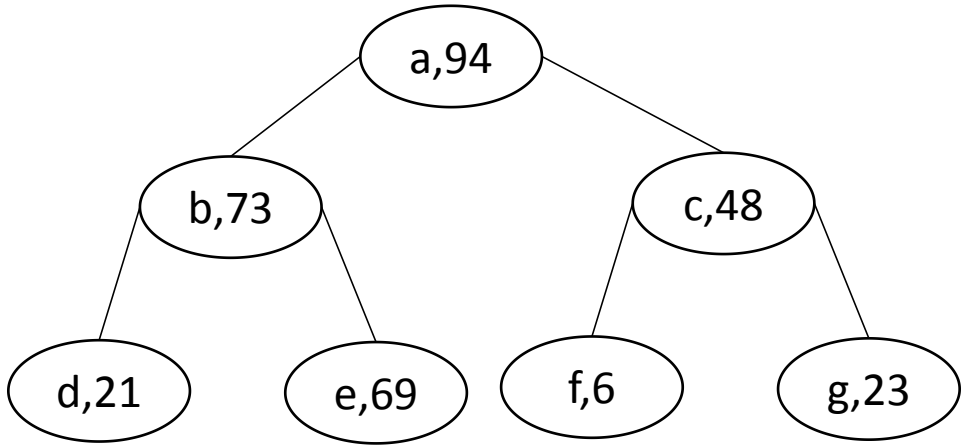
Application 4 : File d'attente avec priorité

La file est représentée par un arbre binaire particulier
Un noeud = (Information + Priorité)

Tous les niveaux de l'arbre sont pleins (sauf éventuellement le dernier)

La racine contient l'élément avec la plus forte priorité

Chaque nœud interne à une priorité supérieure à celle des ses fils

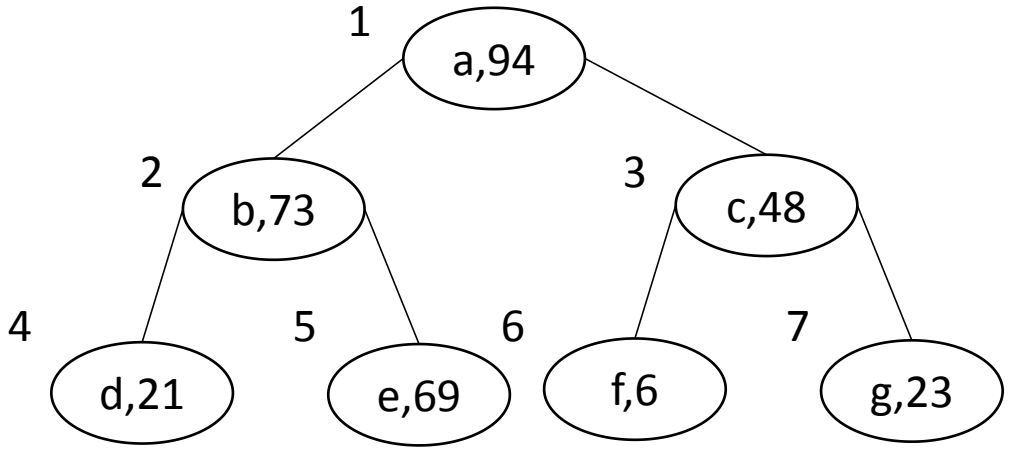


Les arbres binaires

Application 4 : File d'attente avec priorité

Les noeuds sont numérotés: le noeud à la position P est le père des noeuds aux positions 2P et 2P+1

Pour aller au noeud à la position P
P = $b_1 b_2 b3 .... b_n$. (Représentation binaire)
Pour i=2, n
  Si b_i = 1 aller à gauche sinon allera a droite Fsi
Fpour



Pour aller au noeud de position 6
6 = **1 1 0**
De la racine, allera à droite puis à gauche

Les arbres binaires

Application 4 : File d'attente avec priorité (Opérations)

Enfiler(v) =

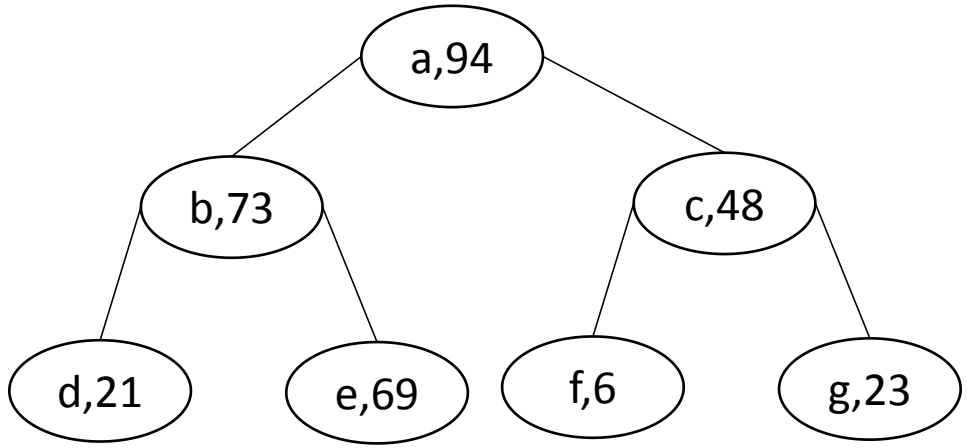
- Ajouter un nœud au dernier niveau, le plus à droite
- Permuter avec ses ascendants jusqu'à ce que sa priorité soit inférieure à celle de son père

→ O(log n)

Défiler(v) =

- Retirer la racine et la remplacer par le nœud le plus à droite du dernier niveau.
- Permuter avec ses descendants jusqu'à ce que sa priorité soit supérieure à celle de ses 2 fils

→ O(log n)



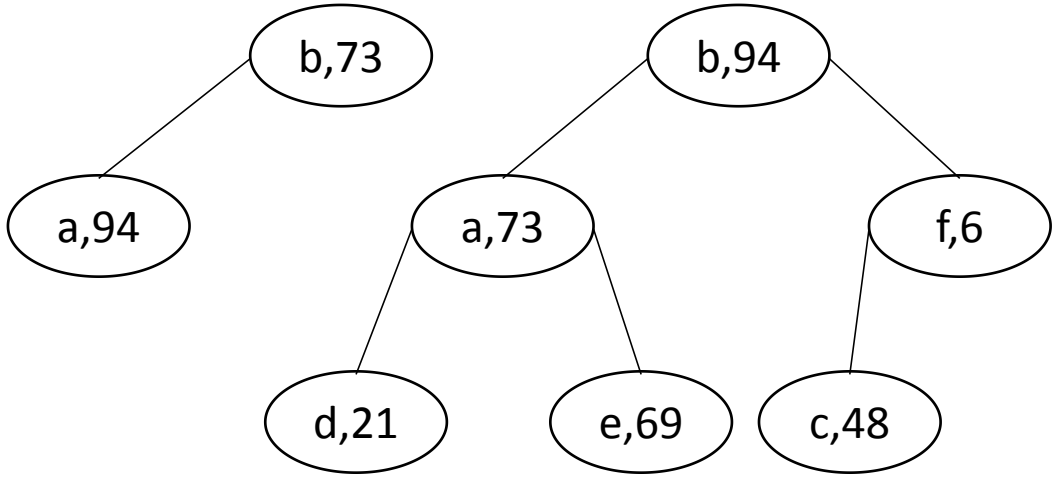
Meilleure qu'une implémentation avec une liste linéaire chaînée ordonnée

Les arbres binaires

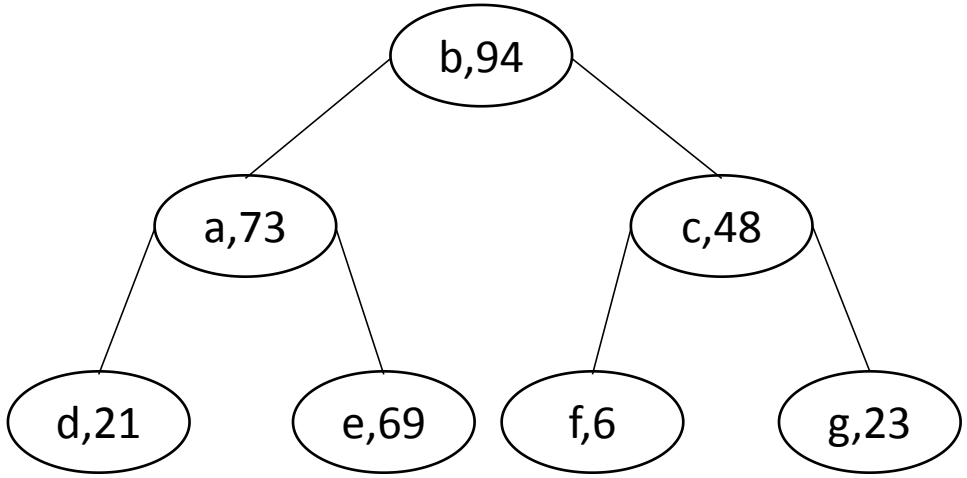
Application 4 : File d'attente avec priorité (Exemple: Enfilement)

b,73 a,94 f,6 d,21 e,69 c,48 g,23

Nb_heap	1	2	3	4	5	6	7
Binaire		10	11	100	101	110	111



Permutation



Permutation

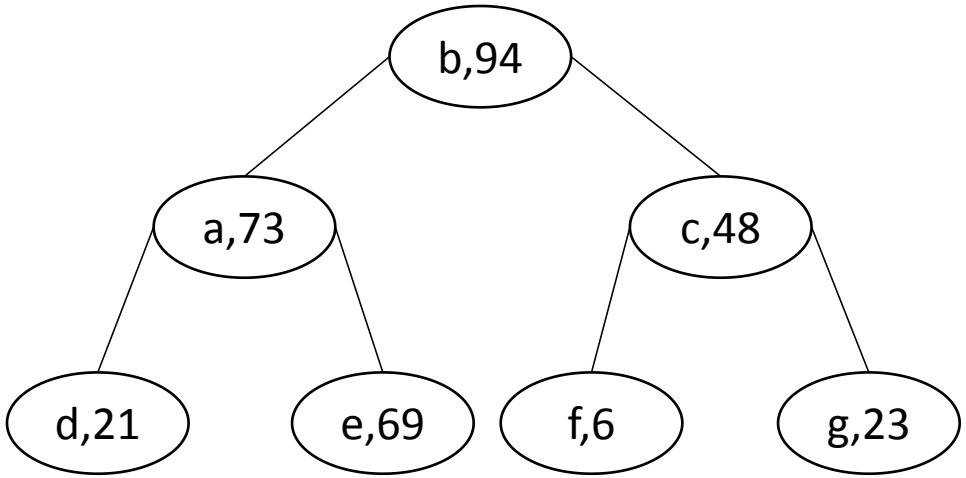
Les arbres binaires

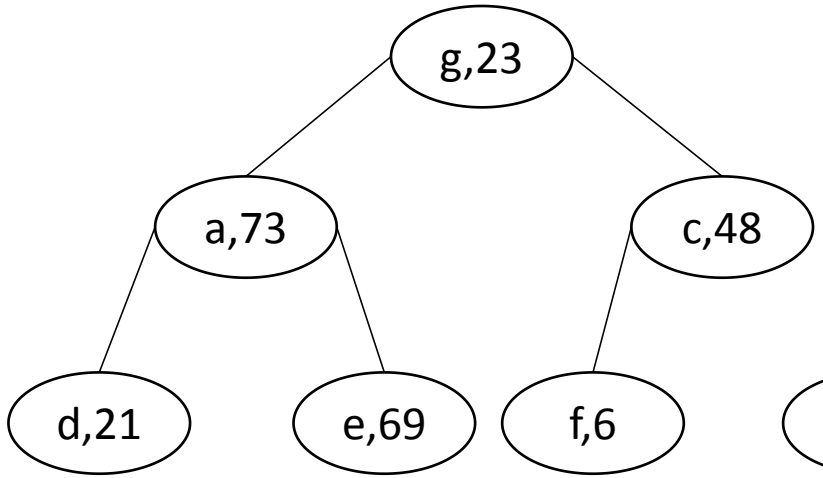
Application 4 : File d'attente avec priorité (Exemple: défilement)

Nb_heap 7

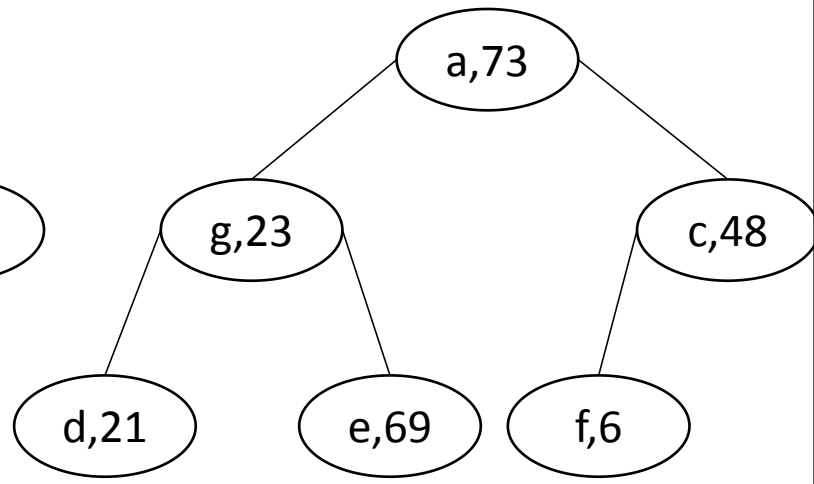
Binaire 111

b





Permutation



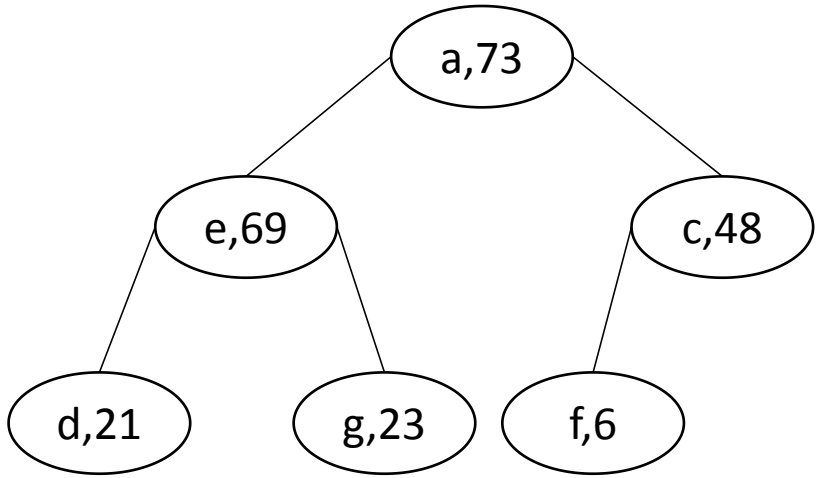
Permutation

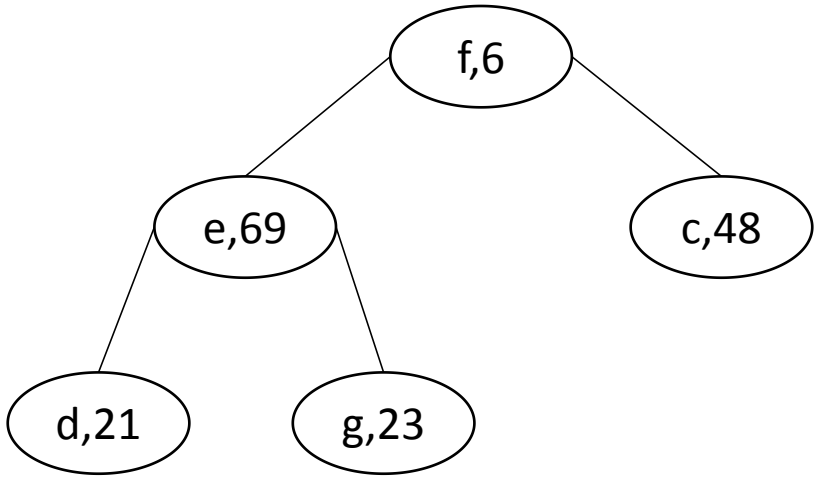
Les arbres binaires

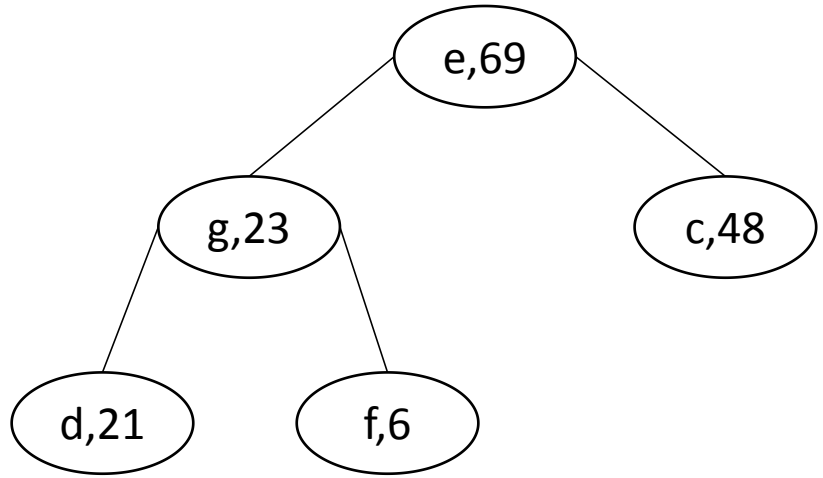
Application 4 : File d'attente avec priorité (Exemple: défilement)

b a

Nb_heap	7	6
Binaire	111	110







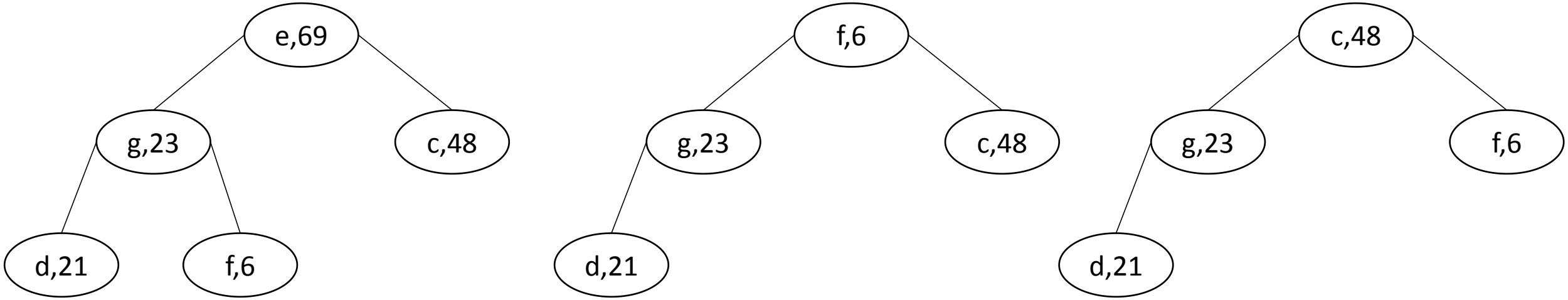
2 permutations

Les arbres binaires

Application 4 : File d'attente avec priorité (Exemple: Défilement)

b a e

Nb_heap	7	6	5
Binaire	111	110	101



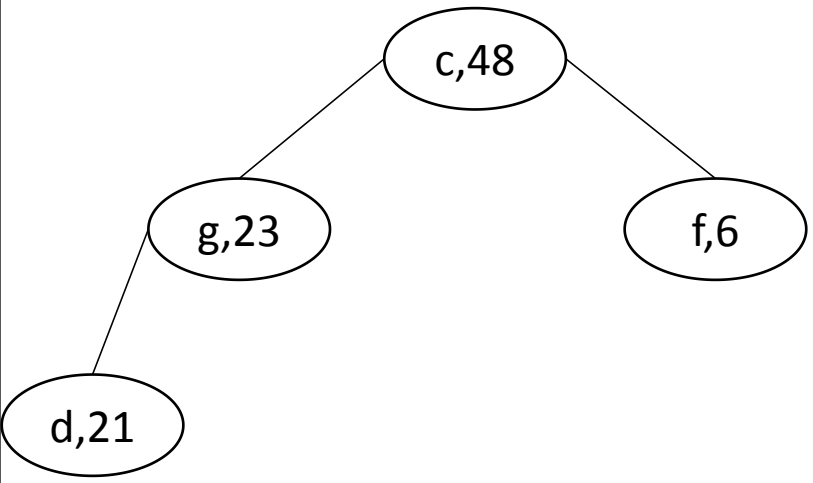
Permutation

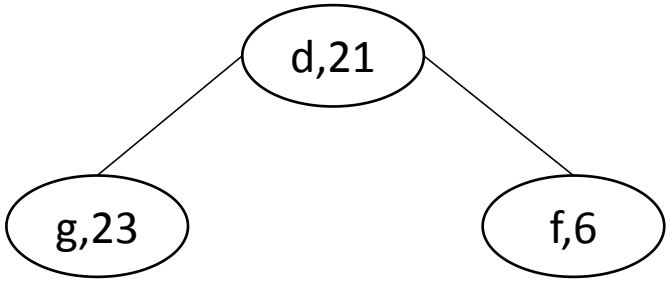
Les arbres binaires

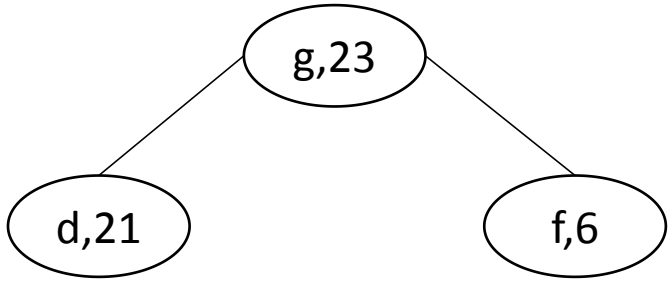
Application 4 : File d'attente avec priorité (Exemple: Défilement)

b a e c

Nb_heap	7	6	5	4
Binaire	111	110	101	100







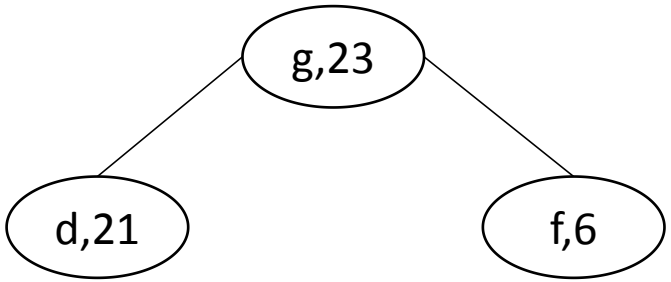
Permutation

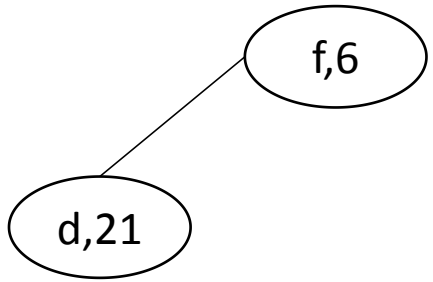
Les arbres binaires

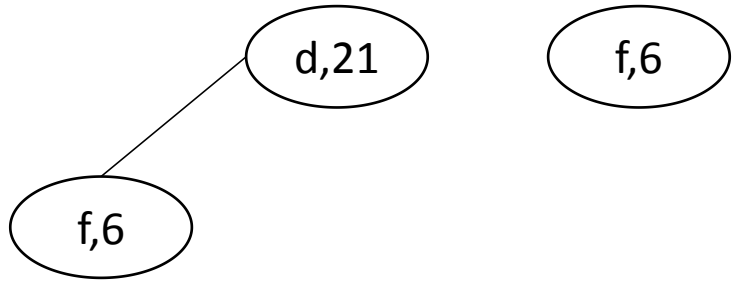
Application 4 : File d'attente avec priorité (Exemple: Défilement)

b a e c g d f

Nb_heap	7	6	5	4	3	2	1	0
Binaire	111	110	101	100	11	10		







Permutation

Les arbres binaires

Application 5 : Code de Huffmann

Soit A un alphabet avec les caractères a, b, c, d, e apparaissant avec des probabilités donnés

Caractère	Probabilité
a	0.12
b	0.4
c	0.15
d	0.08
e	0.25

Coder chaque caractère en une suite de bits avec la propriété du préfixe (Aucun code n'est la préfixe d'un autre code

La propriété du préfixe assure l'opération de décodage

Les arbres binaires

Application 5 : Code de Huffmann

Les 2 codes ont la propriété du préfixe

Pour le code 2, la séquence 1101001 est décodée en bcd.

Principe de décodage:
1. Prendre successivement une séquence de 1 bit, de 2 bits, ... et voir si elle correspond à un code
2. Si oui la décoder et la supprimer de la séquence d'entrée.

Répéter 1 et 2 jusqu'à epuisement de la séquence d'entrée.

Caractère	Probabilité	Code1	Code2
a	0.12	000	000
b	0.4	001	11
c	0.15	010	01
d	0.08	011	001
e	0.25	100	10

1 1 0 1 0 0 1

b c d

Scénario: décodage

Les arbres binaires

Application 5 : Code de Huffmann

Comment comparer les 2 codes?

Calcul de la longueur moyenne

Code 1 :

Longueur moyenne = 3

Code 2 :

Longueur moyenne = 2.22

Longueur moyenne = Somme (L_i*P_i), i=1, 5

L_i : longueur du code du i-ème caractère

P_i : probabilité d'apparition du i-ème caractère

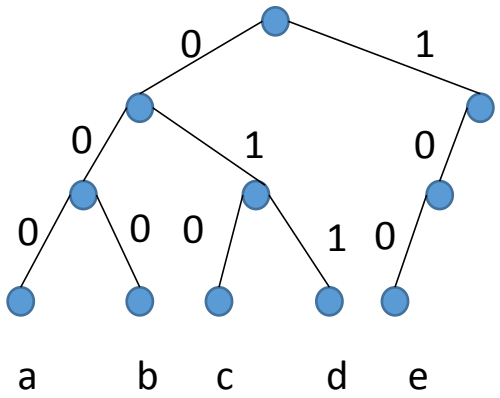
Caractère	Probabilité	Code1	Code2
a	0.12	000	000
b	0.4	001	11
c	0.15	010	01
d	0.08	011	001
e	0.25	100	10

Comment trouver le meilleur code ?

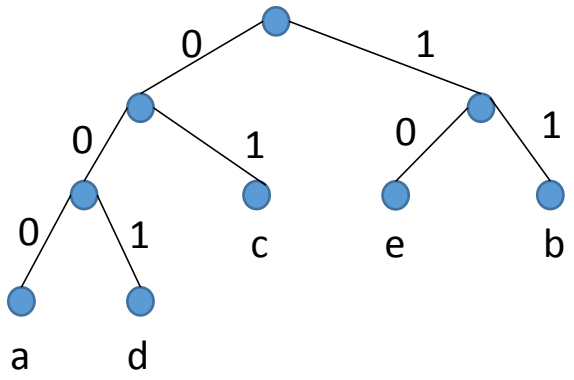
Les arbres binaires

Application 5 : Code de Huffmann

A tout code, on peut associer un arbre binaire



Code 1



Code 2

Caractère	Probabilité	Code1	Code2
a	0.12	000	000
b	0.4	001	11
c	0.15	010	01
d	0.08	011	001
e	0.25	100	10

Inversement, on peut dessiner un arbre binaire avec exactement 5 feuilles et lui associer un code

Les arbres binaires

Application 5 : Code de Huffmann

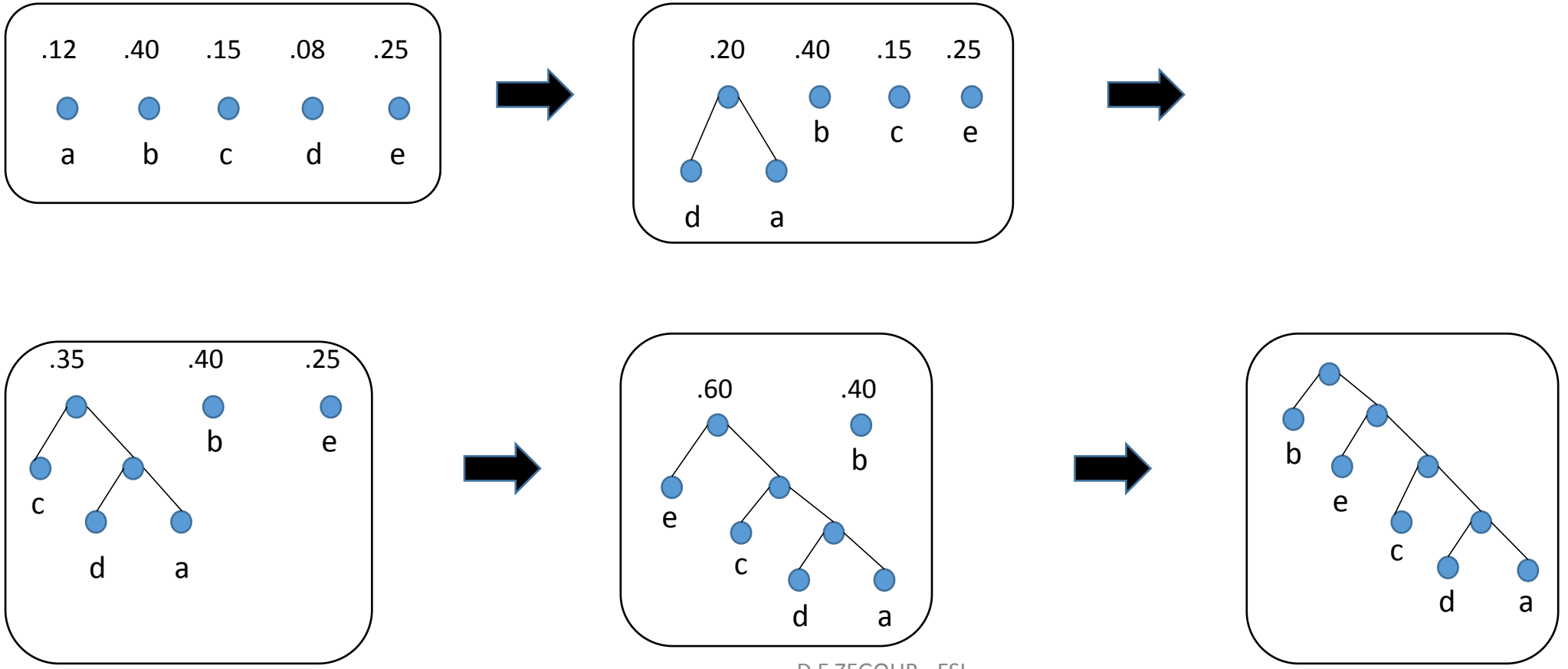
1. Sélectionner deux caractères a et b ayant les plus faibles probabilités.

2. Remplacer les par un caractère fictif x tel que p(x) = p(a) + p(b)

3. Répéter 1. et 2. récursivement.

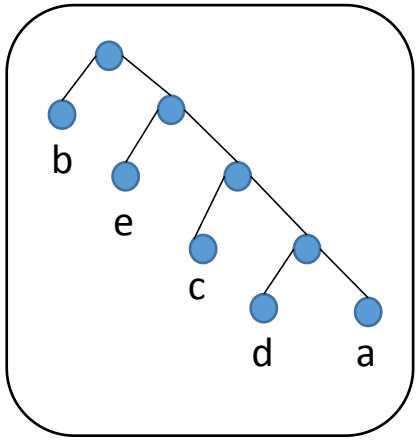
Les arbres binaires

Application 5 : Code de Huffmann



Les arbres binaires

Application 5 : Code de Huffmann



Caractère	Probabilité	Code
a	0.12	1111
b	0.4	1
c	0.15	110
d	0.08	1110
e	0.25	10

Longueur moyenne : 2.17

Les arbres binaires

Implémentation des arbres binaires (dynamique en C)

```
/** Arbres de recherche binaire **/

typedef int Typeelem_Ai   ;
typedef struct Noeud_Ai * Pointeur_Ai ;

struct Noeud_Ai
   {
     Typeelem_Ai  Val ;
     Pointeur_Ai Fg ;
     Pointeur_Ai Fd ;
   }
```

Les arbres binaires

Implémentation des arbres binaires (dynamique en C)

```
void Creernoeud_Ai( Pointeur_Ai *P)
   {
     *P = (struct Noeud_Ai *) malloc( sizeof( struct Noeud_Ai))   ;
     (*P)->Fg = NULL;
     (*P)->Fd = NULL;
   }

void Liberernoeud_Ai( Pointeur_Ai P)
   {  free( P ) ; }
```

Les arbres binaires

Implémentation des arbres binaires (dynamique en C)

```
Typeelem_Ai Info_Ai( Pointeur_Ai P )
   { return P->Val;   }

Pointeur_Ai Fg_Ai( Pointeur_Ai P)
   { return P->Fg ; }

Pointeur_Ai Fd_Ai( Pointeur_Ai P)
   { return P->Fd ; }
```

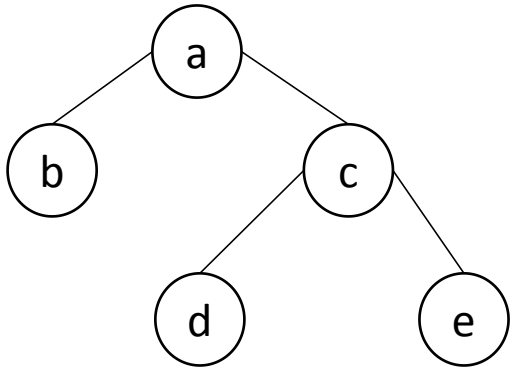
Les arbres binaires

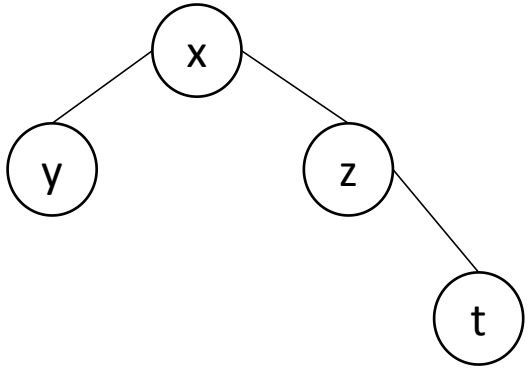
Implémentation des arbres binaires (dynamique en C)

```
void Aff_info_Ai ( Pointeur_Ai P, Typeelem_Ai Val)
   {
      P->Val = Val ;
   }

void Aff_fg_Ai( Pointeur_Ai P, Pointeur_Ai Q)
   { P->Fg =  Q;  }

void Aff_fd_Ai( Pointeur_Ai P, Pointeur_Ai Q)
   { P->Fd =  Q ; }
```

Les arbres binaires

Implémentation des arbres binaires (Statique)





Plusieurs arbres binaires dans un même tableau.

0	3	a	4	V
1	-1	y	-1	V
2	1	x	5	V
3	-1	b	-1	V
4	6	c	8	V
5	-1	z	7	V
6	-1	d	-1	V
7	-1	t	-1	V
8	-1	e	-1	V
				F
				...
Max				F

Les arbres binaires

Implémentation des arbres binaires (Statique)

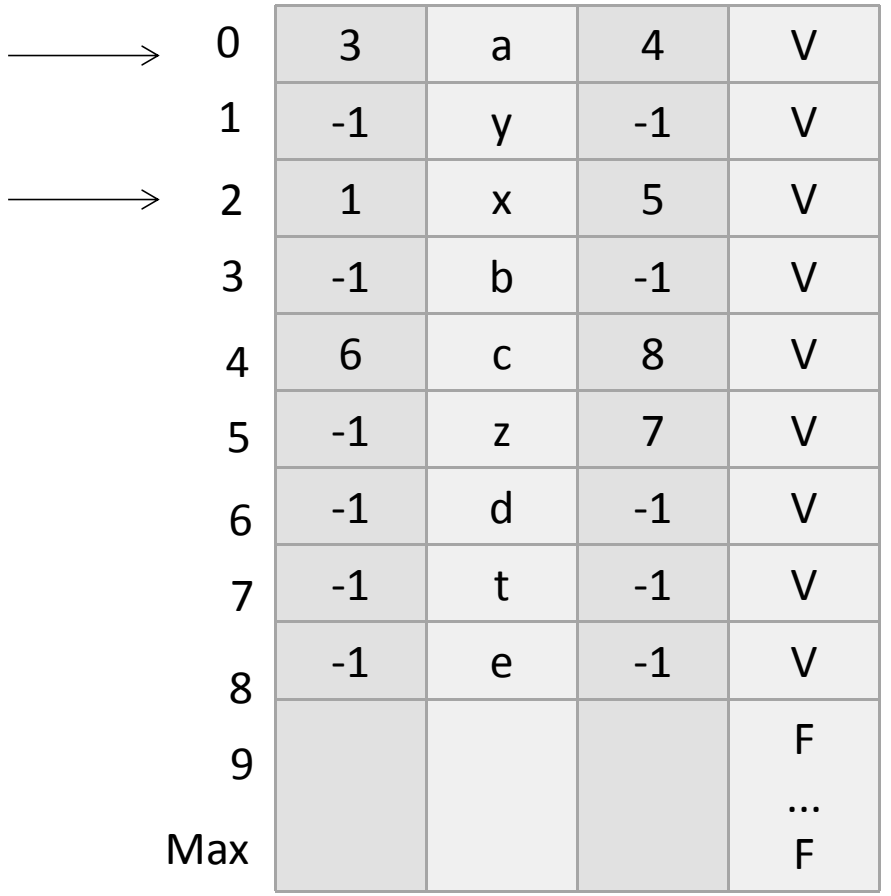
Un élément = (Info, Fg, Fd, Occupe).

Le champ "Occupe" est nécessaire pour les opérations Creernoeud et Liberernoeud.

Une phase d'initialisation est obligatoire avant l'utilisation de ce tableau (Champ Occupe à Faux)

Donc le tableau est global.

Un arbre binaire est défini par l'indice de son premier élément.



0	3	a	4	V
1	-1	y	-1	V
2	1	x	5	V
3	-1	b	-1	V
4	6	c	8	V
5	-1	z	7	V
6	-1	d	-1	V
7	-1	t	-1	V
8	-1	e	-1	V
9				F
				...
Max				F

33

Les arbres binaires

Implémentation des arbres binaires (Statique en C)

```
#define Max 100
#define True 1
#define False 0
#define Nil -1
typedef int Bool;
typedef int Typeqq;
struct Typearb
 {
   Typeqq Info ;
   int Fg;
   int Fd;
   Bool Occupe;
 };
```

```
/* Initialisation */
 void Init()
  {
    int I;
    for (I=0; I<Max; I++)
      Arb[I].Occupe = False;
  }
```

Les arbres binaires

Implémentation des arbres binaires (Statique en C)

```
void Creernoeud ( int *I )
  {
   Bool Trouv;
   *I = 0;
   Trouv = False;
   while ( *I < Max && !Trouv )
    if ( Arb[*I].Occupe )
      *I++  ;
    else
      Trouv = True;
   if  ( !Trouv )  *I = -1;
  }
```

```
void Liberernoeud ( int I )
 {
   Arb[I].Occupe = False ;
 }
```

Les arbres binaires

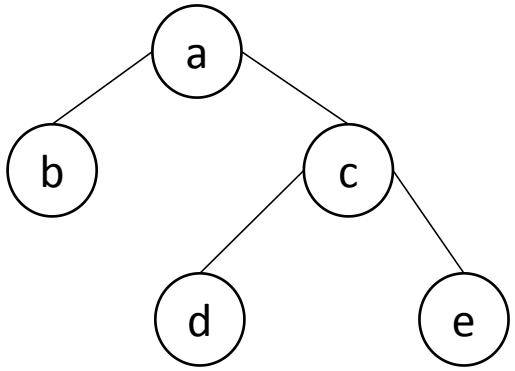
Implémentation des arbres binaires (Statique en C)

```
Typeqq Info ( int I )
  {
    return( Arb[I].Info );
  }

int Fd ( int I )
  {
    return ( Arb[I].Fd ) ;
  }

int Fg ( int I )
  {
    return ( Arb[I].Fg ) ;
  }
```

```
void Aff_info ( int I, Typeqq Val)
  {
    Arb[I].Info = Val;
  }

void Aff_fd ( int I, int J)
  {
    Arb[I].Fd = J;
  }

void Aff_fg ( int I, int J)
  {
    Arb[I].Fg = J;
  }
```

Les arbres binaires

Implémentation des arbres binaires (Représentation séquentielle)



Idée : Ne pas représenter les indices des fils.

a	b	c			d	e	
1	2	3	4	5	6	7	Max

La racine est à la position 1.

Le noeud à la position p est le père implicite des noeuds 2p (fils gauche) et 2p+1 (fils droit).

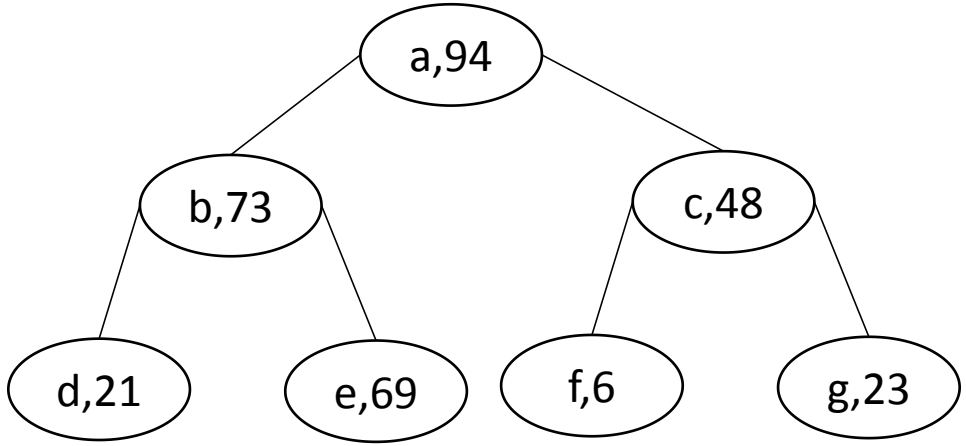
Si un fils gauche est à la position p, son frère droit est à la position p+1;

Si un fils droit est à la position p son frère gauche est la position p-1.

Père(p) c'est le noeud p DIV 2.

Les arbres binaires

Implémentation des arbres binaires (Représentation séquentielle: application : File d'attente avec priorité)



a,94	b,73	c,48	d,21	e,69	f,6	g,23	.....
1	2	3	4	5	6	7	Max

Enfiler(F, v) : n := n+ 1, T[n] := v ; Permutations éventuelles
(pour avoir le père c'est P Div 2)

Défiler (F, v) : T[1] := T[n]; n:=n - 1 ; Permutations éventuelles
(Pour avoir les fils gauche et droit, c'est 2P et 2P+1 respectivement