

Arbres quaternaires

Algorithmique et structures de données, 2022-2023

P. Albuquerque (B410), P. Künzli et O. Malaspinas (A401), ISC, HEPIA
2023-04-28

En partie inspirés des supports de cours de P. Albuquerque

Les arbres quaternaires

Définition

Arbre dont chaque nœud a 4 enfants ou aucun.

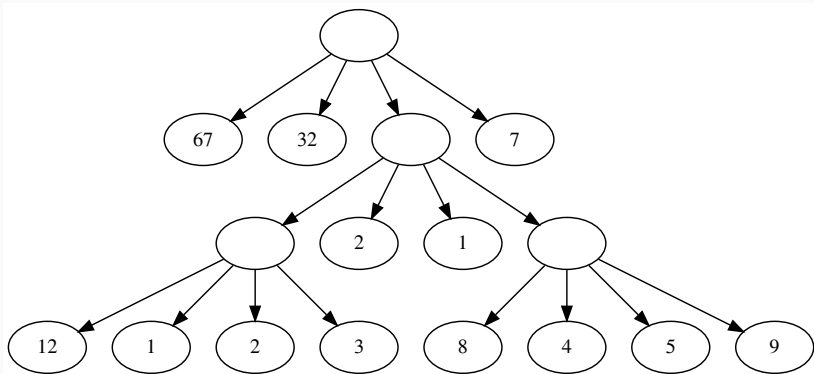


Figure 1: Un exemple d'arbre quaternaire.

Cas d'utilisation

Typiquement utilisés pour représenter des données bidimensionnelles.

Son équivalent tri-dimensionnel est l'octree (chaque nœud a 8 enfants ou aucun).

Cas d'utilisation: images

- Stockage: compression.
- Transformations: symétries, rotations, etc.

Cas d'utilisation: simulation

- Indexation spatiale.
- Détection de collisions.
- Simulation de galaxies, Barnes-Hut.

Exemple de compression

Comment représenter l'image

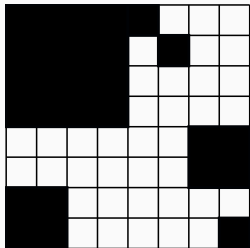


Figure 2: Image noir/blanc.

Sous la forme d'un arbre quaternaire?

Exemple de compression

Comment représenter l'image

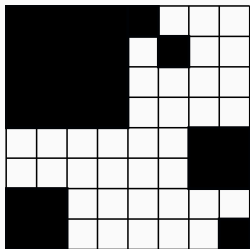


Figure 2: Image noir/blanc.

Sous la forme d'un arbre quaternaire?

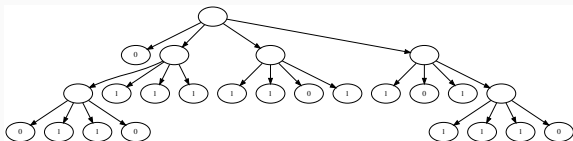


Figure 3: L'arbre quaternaire correspondant.

Économie?

Exemple de compression

Comment représenter l'image

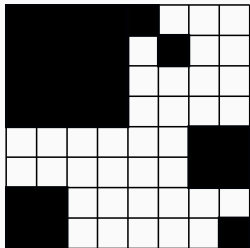


Figure 2: Image noir/blanc.

Sous la forme d'un arbre quaternaire?

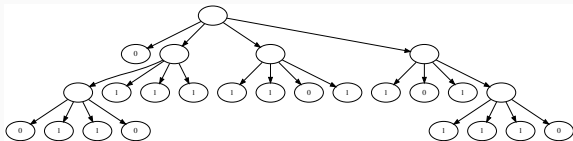


Figure 3: L'arbre quaternaire correspondant.

Économie?

Image 64 pixels, arbre 25 nœuds.

Pseudo-code?

Pseudo-code?

```
struct node
  info
  node sup_gauche, sup_droit,
    inf_gauche, inf_droit
```

En C?

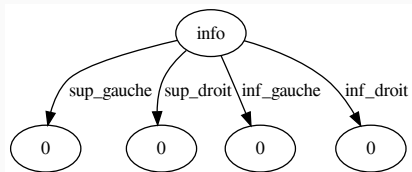


Figure 4: Un nœud d'arbre quaternaire.

Structure de données

Pseudo-code?

```
struct node
    info
    node sup_gauche, sup_droit,
        inf_gauche, inf_droit
```

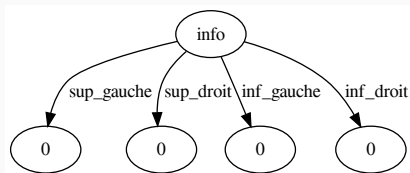


Figure 4: Un nœud d'arbre quaternaire.

En C?

```
struct _node {
    int info;
    struct _node *sup_left;
    struct _node *sup_right;
    struct _node *inf_left;
    struct _node *inf_right;
};
```

- Pourquoi le * est important?

Structure de données

Pseudo-code?

```
struct node
    info
    node sup_gauche, sup_droit,
        inf_gauche, inf_droit
```

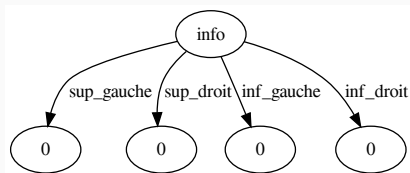


Figure 4: Un nœud d'arbre quaternaire.

En C?

```
struct _node {
    int info;
    struct _node *sup_left;
    struct _node *sup_right;
    struct _node *inf_left;
    struct _node *inf_right;
};
```

- Pourquoi le * est important?
- Type récursif => taille inconnue à la compilation.

Une fonctionnalité simple

La fonction `est_feuille`(noeud)

- Problème avec cette implémentation?

```
bool est_feuille(noeud)
    retourne
        est_vider(sup_gauche(noeud)) &&
        est_vider(sup_droit(noeud)) &&
        est_vider(inf_gauche(noeud)) &&
        est_vider(inf_droit(noeud))
```

Une fonctionnalité simple

La fonction `est_feuille`(noeud)

- Problème avec cette implémentation?

```
bool est_feuille(noeud)
    retourne
        est_vider(sup_gauche(noeud)) &&
        est_vider(sup_droit(noeud)) &&
        est_vider(inf_gauche(noeud)) &&
        est_vider(inf_droit(noeud))
```

- Inutile d'avoir 4 conditions (soit 4 enfants soit aucun!)
- Facile d'en oublier un!
- Comment changer la structure pour que ça soit moins terrible?

Une fonctionnalité simple

La fonction `est_feuille`(noeud)

- Problème avec cette implémentation?

```
bool est_feuille(noeud)
    retourne
        est_vide(sup_gauche(noeud)) &&
        est_vide(sup_droit(noeud)) &&
        est_vide(inf_gauche(noeud)) &&
        est_vide(inf_droit(noeud))
```

- Inutile d'avoir 4 conditions (soit 4 enfants soit aucun!)
- Facile d'en oublier un!
- Comment changer la structure pour que ça soit moins terrible?

```
struct node
    info
    node enfant[4]
```

En C?

Structure de données

En C?

```
typedef struct _node {  
    int info;  
    struct _node *child[4];  
} node;
```

Fonction `is_leaf(node *tree)?`

Structure de données

En C?

```
typedef struct _node {
    int info;
    struct _node *child[4];
} node;
```

Fonction is_leaf(node *tree)?

```
bool is_leaf(node *tree) {
    return (NULL == tree->child[0]); // only first matters
}
```


Problème à résoudre

- Construire un arbre quaternaire à partir d'une image:
 - Créer l'arbre (allouer la mémoire pour tous les nœuds),
 - Le remplir avec les valeurs des pixels.
- Compression de l'image:
 - Si les pixels sont les mêmes dans le quadrant on supprime le sous-arbre (sans perte)
 - Si les pixels dévient pas trop on supprime le quadrant (avec perte)

Création de l'arbre

Comment créer un arbre de profondeur `prof` (3min)?

Création de l'arbre

Comment créer un arbre de profondeur prof (3min)?

```
arbre creer_arbre(prof)
    n = nouveau_noeud() # alloue la mémoire
    si prof > 0
        pour i = 0 à 3
            n.enfant[i] = creer_arbre(prof-1)
    retourne n
```

En C (3 min, matrix)?

Création de l'arbre

Comment créer un arbre de profondeur prof (3min)?

```
arbre creer_arbre(prof)
  n = nouveau_noeud() # alloue la mémoire
  si prof > 0
    pour i = 0 à 3
      n.enfant[i] = creer_arbre(prof-1)
  retourne n
```

En C (3 min, matrix)?

```
node *qt_create(int depth) {
  node *n = calloc(1, sizeof(node));
  if (depth > 0) {
    for (int i = 0; i < 4; ++i) {
      n->child[i] = qt_create(depth-1);
    }
  }
  return n;
}
```

Le nombre de nœuds?

Comment implémenter la fonction (pseudo-code, 5min, matrix)?

Le nombre de nœuds?

Comment implémenter la fonction (pseudo-code, 5min, matrix)?

```
entier nombre_nœuds(arbre)
  si est_feuille(arbre)
    retourne 1
  sinon
    somme = 1
    pour i de 0 à 3
      somme += nombre_nœuds(arbre.enfant[i])
    retourne somme
```

Le nombre de nœuds?

Comment implémenter la fonction en C (3min, matrix)?

Le nombre de nœuds?

Comment implémenter la fonction en C (3min, matrix)?

```
int size(node *qt) {
    if (is_leaf(qt)) {
        return 1;
    } else {
        int sum = 1;
        for (int i = 0; i < 4; ++i) {
            sum += size(qt->child[i]);
        }
        return sum;
    }
}
```


La profondeur en C?

Implémentation (5min, matrix)

La profondeur en C?

Implémentation (5min, matrix)

```
int max(int x, int y) {
    return (x >= y ? x : y);
}

int max_depth(int depths[4]) {
    int m = depths[0];
    for (int i = 1; i < 4; ++i) {
        m = max(m, depths[i]);
    }
    return m;
}

int depth(node *qt) {
    int depths[] = {0, 0, 0, 0};
    if (is_leaf(qt)) {
        return 0;
    } else {
        for (int i = 0; i < 4; ++i) {
            depths[i] = depth(qt->child[i]);
        }
        return 1 + max_depth(depths);
    }
}
```

Fonctions utiles (1/4)

Comment remplir un arbre depuis une matrice?

SG=0 | SD=1

21 | 12 | 4 | 4

9 | 7 | 4 | 4

1 | 1 | 0 | 31

1 | 1 | 3 | 27

IG=2 | ID=3

Quel arbre cela représente?

Fonctions utiles (1/4)

Comment remplir un arbre depuis une matrice?

SG=0 | SD=1

21 | 12 | 4 | 4

9 | 7 | 4 | 4

1 | 1 | 0 | 31

1 | 1 | 3 | 27

IG=2 | ID=3

Quel arbre cela représente?

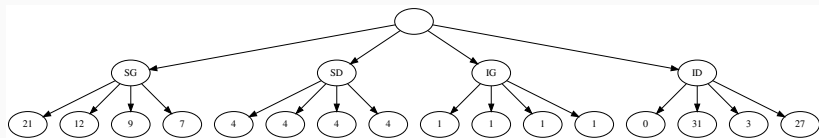


Figure 5: L'arbre correspondant

Fonctions utiles (2/4)

- On veut transformer une ligne/colonne en feuille.
- Comment?

Soit ligne=2, colonne=3

SG=0		SD=1	
21		12	4 4
9		7	4 4

1		1	0 31
1		1	3 27
IG=2		ID=3	

Trouver un algorithme

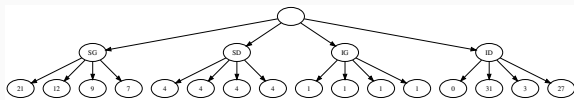


Figure 6: Déterminer un algorithme.

- Quelle feuille pour 31 (li=2, co=3)?
- Plus important: quel chemin?

Fonctions utiles (2/4)

- On veut transformer une ligne/colonne en feuille.
- Comment?

Soit ligne=2, colonne=3

SG=0		SD=1	
21		12	4 4
9		7	4 4

1		1	0 31
1		1	3 27
IG=2		ID=3	

Trouver un algorithme

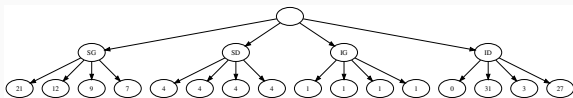


Figure 6: Déterminer un algorithme.

- Quelle feuille pour 31 (li=2, co=3)?
- Plus important: quel chemin?
- co -> G/D, li -> S/I,
- $2 * (li / 2) + co / 2 \rightarrow 2 * 1 + 1 = 3$
- $2 * ((li \% 2) / 1) + (co \% 2) / 1 \rightarrow 2 * 0 + 1 = 1$
- Comment généraliser?

Fonctions utiles (3/4)

Soit ligne=2, colonne=3

SG=0 | SD=1

21 | 12 | 4 | 4

9 | 7 | 4 | 4

1 | 1 | 0 | 31

1 | 1 | 3 | 27

IG=2 | ID=3

Trouver un algorithme (prendre plusieurs exemples, 15min, matrix)

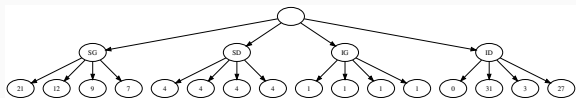


Figure 7: Déterminer un algorithme.

- Comment généraliser?

Fonctions utiles (3/4)

Soit ligne=2, colonne=3

SG=0 | SD=1

21 | 12 | 4 | 4

9 | 7 | 4 | 4

1 | 1 | 0 | 31

1 | 1 | 3 | 27

IG=2 | ID=3

Trouver un algorithme (prendre plusieurs exemples, 15min, matrix)

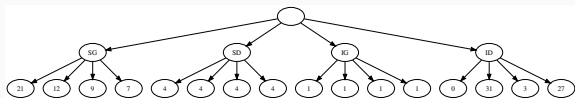


Figure 7: Déterminer un algorithme.

- Comment généraliser?

```
noeud position(li, co, arbre)
d = profondeur(arbre);
tant_que (d >= 1)
    index = 2 * ((li % 2^d) / 2^(d-1))
            (col % 2^d) / 2^(d-1)
    arbre = arbre.enfant[index]
    d -= 1
retourne arbre
```


Fonctions utiles (4/4)

Pseudo-code

```
noeud position(li, co, arbre)
  d = profondeur(arbre);
  tant_que (d >= 1)
    index = 2 * ((li % 2^d) / 2^(d-1)) +
            (col % 2^d) / 2^(d-1)
    arbre = arbre.enfant[index]
    d -= 1
  retourne arbre
```

Écrire le code C correspondant (5min, matrix)

Remplir l'arbre

A partir d'une matrice (pseudo-code, 5min, matrix)?

Remplir l'arbre

A partir d'une matrice (pseudo-code, 5min, matrix)?

```
arbre matrice_à_arbre(matrice)
    arbre = creer_arbre(profondeur)
    pour li de 0 à nb_lignes(matrice)
        pour co de 0 à nb_colonnes(matrice)
            noeud = position(li, co, arbre)
            noeud.info = matrice[co][li]
    retourne arbre
```

Remplir l'arbre

A partir d'une matrice (pseudo-code, 5min, matrix)?

```
arbre matrice_à_arbre(matrice)
    arbre = creer_arbre(profondeur)
    pour li de 0 à nb_lignes(matrice)
        pour co de 0 à nb_colonnes(matrice)
            noeud = position(li, co, arbre)
            noeud.info = matrice[co][li]
    retourne arbre
```

A partir d'une matrice (C, 5min, matrix)?

Remplir l'arbre

A partir d'une matrice (pseudo-code, 5min, matrix)?

```
arbre matrice_à_arbre(matrice)
    arbre = creer_arbre(profondeur)
    pour li de 0 à nb_lignes(matrice)
        pour co de 0 à nb_colonnes(matrice)
            noeud = position(li, co, arbre)
            noeud.info = matrice[co][li]
    retourne arbre
```

A partir d'une matrice (C, 5min, matrix)?

```
node *matrix_to_qt(int nb_li, int nb_co, int matrix[nb_li][nb_co], int depth)
{
    node *qt = qt_create(depth);
    for (int li = 0; li < nd_li; ++li) {
        for (int co = 0; co < nd_co; ++co) {
            node *current = position(li, co, qt);
            current->info = matrix[li][co];
        }
    }
    return qt;
}
```

Remplir la matrice

A partir de l'arbre (pseudo-code, 3min, matrix)?

Remplir la matrice

A partir de l'arbre (pseudo-code, 3min, matrix)?

```
matrice arbre_à_matrice(arbre)
    matrice = creer_matrice(nb_lignes(arbre), nb_colonnes(arbre))
    pour li de 0 à nb_lignes(matrice)
        pour co de 0 à nb_colonnes(matrice)
            noeud = position(li, co, arbre)
            matrice[co][li] = noeud.info
    retourne matrice
```

Remplir la matrice

A partir de l'arbre (pseudo-code, 3min, matrix)?

```
matrice arbre_à_matrice(arbre)
    matrice = creer_matrice(nb_lignes(arbre), nb_colonnes(arbre))
    pour li de 0 à nb_lignes(matrice)
        pour co de 0 à nb_colonnes(matrice)
            noeud = position(li, co, arbre)
            matrice[co][li] = noeud.info
    retourne matrice
```

A partir de l'arbre (C, 3min, matrix)?

Remplir la matrice

A partir de l'arbre (pseudo-code, 3min, matrix)?

```
matrice arbre_à_matrice(arbre)
    matrice = creer_matrice(nb_lignes(arbre), nb_colonnes(arbre))
    pour li de 0 à nb_lignes(matrice)
        pour co de 0 à nb_colonnes(matrice)
            noeud = position(li, co, arbre)
            matrice[co][li] = noeud.info
    retourne matrice
```

A partir de l'arbre (C, 3min, matrix)?

```
void qt_to_matrix(node *qt, int nb_li, int nb_co, int matrix[nb_li][nb_co])
    for (int li = 0; li < nd_li; ++li) {
        for (int co = 0; co < nd_co; ++co) {
            node *current = position(li, co, qt);
            matrix[li][co] = current->info;
        }
    }
```