Arbres

Algorithmique et structures de données, 2023-2024

P. Kunzli (Cloud) et O. Malaspinas (A401), ISC, HEPIA

2024-03-12

En partie inspirés des supports de cours de P. Albuquerque

Rappel: arbre binaire

Qu'est-ce qu'un arbre binaire?

Rappel: arbre binaire

Qu'est-ce qu'un arbre binaire?

- Structure de données abstraite,
- Chaque nœud a au plus deux enfants: gauche et droite,
- Chaque enfants est un arbre.

Rappel: parcous (infixe, GRD)

Rappel: parcous (infixe, GRD)

```
parcours_infixe(arbre a)
    si est_pas_vide(gauche(a))
       parcours_infixe(gauche(a))
    visiter(a)
    si est_pas_vide(droite(a))
       parcours_infixe(droite(a))
```

Rappel: parcours (postfixe, GDR)

Rappel: parcours (postfixe, GDR)

```
parcours_postfixe(arbre a)
    si est_pas_vide(gauche(a))
       parcours_postfixe(gauche(a))
    si est_pas_vide(droite(a))
       parcours_postfixe(droite(a))
    visiter(a)
```

Rappel: parcours (préfixe, RGD)

Rappel: parcours (préfixe, RGD)

```
parcours_préfixe(arbre a)
    visiter(a)
    si est_pas_vide(gauche(a))
        parcours_préfixe(gauche(a))
    si est_pas_vide(droite(a))
        parcours_préfixe(droite(a))
```

La recherche dans un arbre binaire

- Les arbres binaires peuvent retrouver une information très rapidement.
- À quelle complexité? À quelle condition?

La recherche dans un arbre binaire

- Les arbres binaires peuvent retrouver une information très rapidement.
- À quelle complexité? À quelle condition?

Condition

- Le contenu de l'arbre est **ordonné** (il y a une relation d'ordre (<, >) entre les éléments).

Complexité

- La profondeur de l'arbre (ou le $\mathcal{O}(\log_2(N))$)

La recherche dans un arbre binaire

- Les arbres binaires peuvent retrouver une information très rapidement.
- À quelle complexité? À quelle condition?

Condition

- Le contenu de l'arbre est **ordonné** (il y a une relation d'ordre (<, >) entre les éléments).

Complexité

- La profondeur de l'arbre (ou le $\mathcal{O}(\log_2(N))$)

Exemple: les arbres lexicographiques

- Chaque nœud contient une information de type ordonné, la **clé**,
- Par construction, pour chaque nœud N:
  - Toutes clé du sous-arbre à gauche de N sont inférieurs à la clé de N.
  - Toutes clé du sous-arbre à droite de N sont inférieurs à la clé de N.

Algorithme de recherche

- Retourner le nœud si la clé est trouvée dans l'arbre.

```
arbre recherche(clé, arbre)
    tante_que est_non_vide(arbre)
        si clé < clé(arbre)
            arbre = gauche(arbre)
        sinon si clé > clé(arbre)
            arbre = droite(arbre)
        sinon
            retourne arbre
    retourne NULL
```

Algorithme de recherche, implémentation (live)

Algorithme de recherche, implémentation (live)

```c
typedef int key_t;
typedef struct _node {
    key_t key;
    struct _node* left;
    struct _node* right;
} node;
node * search(key_t key, node * tree) {
    node * current = tree;
    while (NULL != current) {
        if (current->key > X) {
            current = current->gauche;
        } else if (current->key < X){
            current = current->droite;
        } else {
            return current;
        }
    }
    return NULL;
}
```

Exercice (5-10min)

Écrire le code de la fonction

```
int tree_size(node * tree);
```

qui retourne le nombre total de nœuds d'un arbre et poster le résultat sur matrix.

Indication: la taille, est 1 + le nombre de nœuds du sous-arbre de gauche additionné au nombre de nœuds dans le sous-arbre de droite.

Exercice (5-10min)

Écrire le code de la fonction

```
int tree_size(node * tree);
```

qui retourne le nombre total de nœuds d'un arbre et poster le résultat sur matrix.

Indication: la taille, est 1 + le nombre de nœuds du sous-arbre de gauche additionné au nombre de nœuds dans le sous-arbre de droite.

```
int arbre_size(node * tree) {
    if (NULL == tree) {
        return 0;
    } else {
        return 1 + tree_size(tree->left)
            + tree_size(tree->right);
    }
}
```

L'insertion dans un arbre binaire

- C'est bien joli de pouvoir faire des parcours, recherches, mais si on peut pas construire l'arbre....

Pour un arbre lexicographique

- Rechercher la position dans l'arbre où insérer.
- Créer un nœud avec la clé et le rattacher à l'arbre.

Exemple d'insertions

- Clés uniques pour simplifier.
- Insertion de 5, 15, 10, 25, 2, -5, 12, 14, 11.
- Rappel:
  - Plus petit que la clé courante => gauche,
  - Plus grand que la clé courante => droite.
- Faisons le dessins ensemble

Exercice (3min, puis matrix)

- Dessiner l'arbre en insérant 20, 30, 60, 40, 10, 15, 25, -5

Pseudo-code d'insertion (1/4)

- Deux parties:
  - Recherche le parent où se passe l'insertion.
  - Ajout de l'enfant dans l'arbre.

Recherche du parent

```
arbre position(arbre, clé)
    si est_non_vide(arbre)
        si clé < clé(arbre)
            suivant = gauche(arbre)
        sinon
            suivant = droite(arbre)
        tant que clé(arbre) != clé && est_non_vide(suivant)
            arbre = suivant
            si clé < clé(arbre)
                suivant = gauche(arbre)
            sinon
                suivant = droite(arbre)
```

Pseudo-code d'insertion (2/4)

- Deux parties:
  - Recherche de la position.
  - Ajout dans l'arbre.

Ajout de l'enfant

```
ajout(arbre, clé)
    si est_vide(arbre)
        arbre = nœud(clé)
    sinon
        si clé < clé(arbre)
            gauche(arbre) = nœud(clé)
        sinon si clé > clé(arbre)
            droite(arbre) = nœud(clé)
        sinon
            retourne
```

Code d'insertion en C

Recherche du parent (ensemble)

Code d'insertion en C

Recherche du parent (ensemble)

```
node * position(node * tree, key_t key) {
    node * current = tree;
    if (NULL != current) {
        node * subtree = key > current->key ? current->right :
            current->left;
        while (key != current->key && NULL != subtree) {
            current = subtree;
            subtree = key > current->key ? current->right :
            current->left;
        }
    }
    return current;
}
```

L'insertion (3/4)

- Deux parties:
  - Recherche de la position.
  - Ajout dans l'arbre.

Ajout du fils (pseudo-code)

```
rien ajout(arbre, clé)
    si est_vide(arbre)
        arbre = nœud(clé)
    sinon
        arbre = position(arbre, clé)
        si clé < clé(arbre)
            gauche(arbre) = nœud(clé)
        sinon si clé > clé(arbre)
            droite(arbre) = nœud(clé)
        sinon
            retourne
```

L'insertion (4/4)

Ajout du fils (code)

- 2 cas: arbre vide ou pas.
- on retourne un pointeur vers le nœud ajouté (ou `NULL`)

L'insertion (4/4)

Ajout du fils (code)

- 2 cas: arbre vide ou pas.
- on retourne un pointeur vers le nœud ajouté (ou `NULL`)

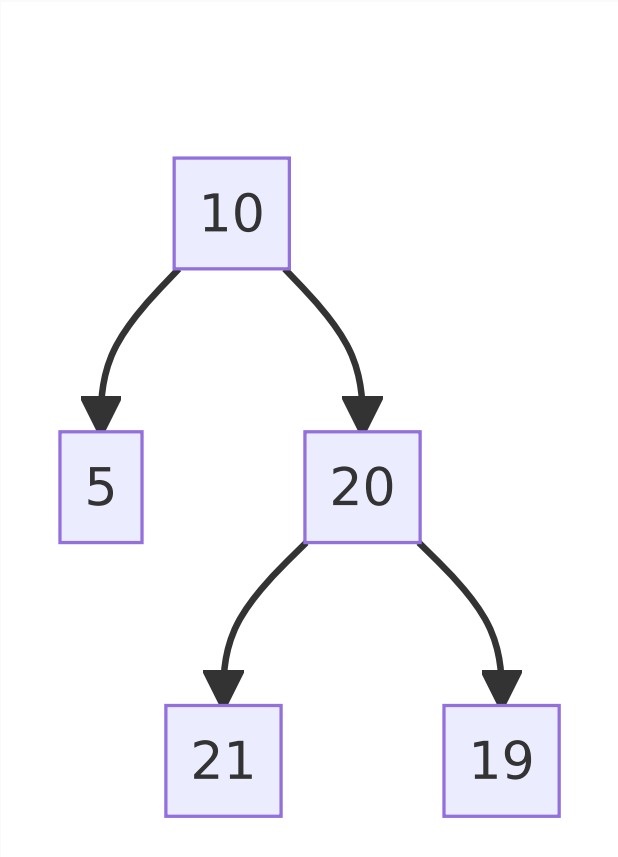
```
node * add_key(node **tree, key_t key) {
    node_t *new_node = calloc(1, sizeof(*new_node));
    new_node->key = key;
    if (NULL == *tree) {
        *tree = new_node;
    } else {
        node * subtree = position(*tree, key);
        if (key == subtree->key) {
            return NULL;
        } else {
            if (key > subtree->key) {
                subtree->right = new_node;
            } else {
                subtree->left = new_node;
            }
        }
    }
    return new_node;
}
```

La suppression de clé

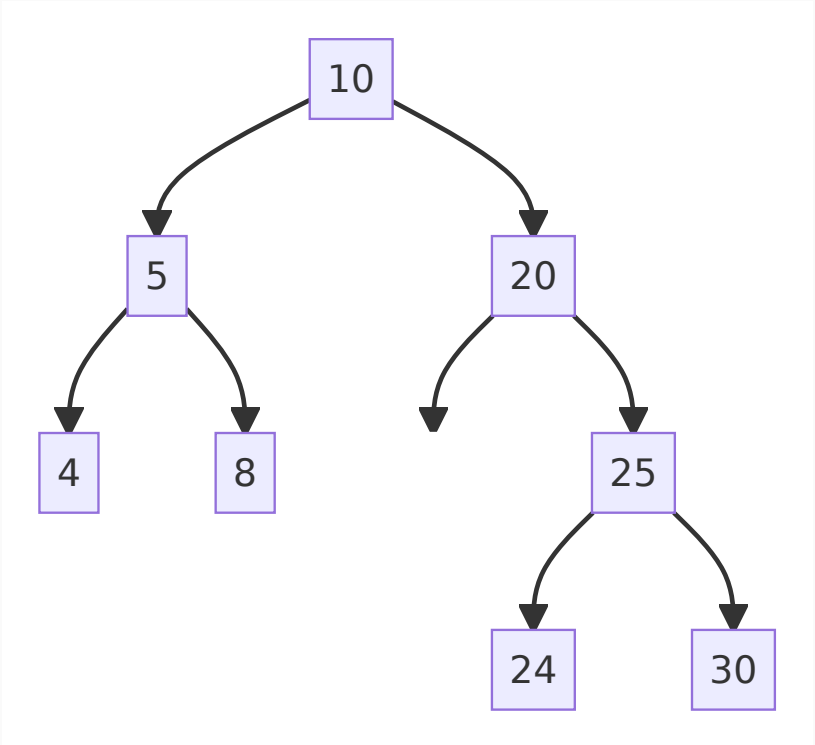
Cas simples:

- le nœud est absent,
- le nœud est une feuille
- le nœuds a un seul fils.

Une feuille (le 19 p.ex.).



Un seul fils (le 20 p.ex.).



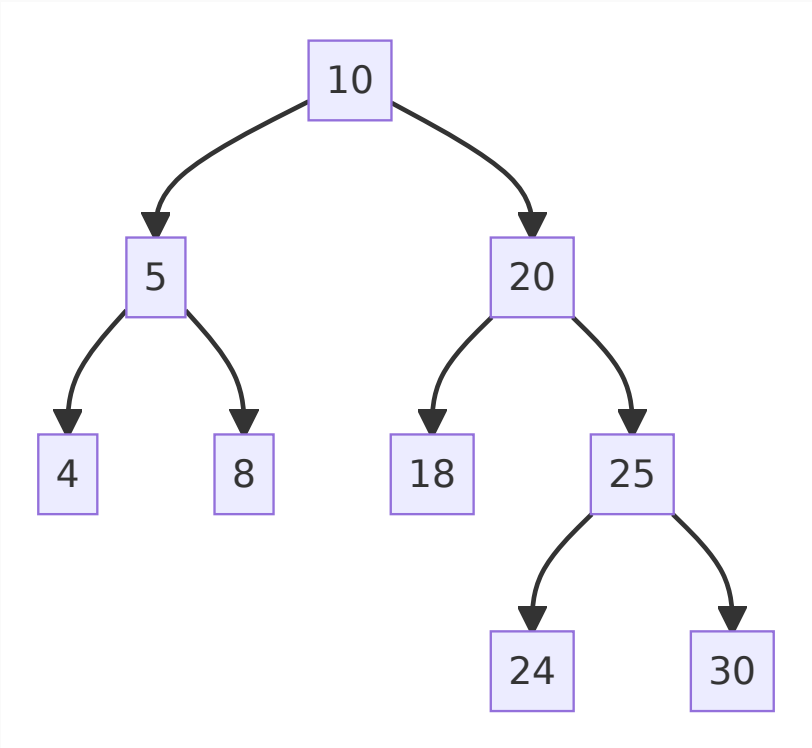
Dans tous les cas

- Chercher le nœud à supprimer: utiliser `position()`.

La suppression de clé

Cas compliqué

- Le nœud à supprimer à (au moins) deux descendants (10).

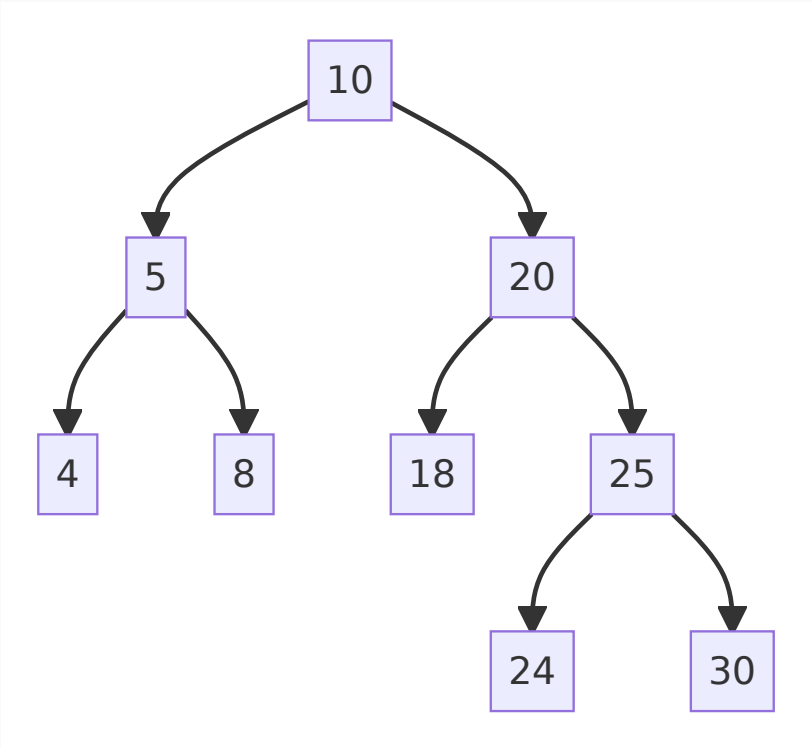


- Si on enlève 10 il se passe quoi?

La suppression de clé

Cas compliqué

- Le nœud à supprimer à (au moins) deux descendants (10).

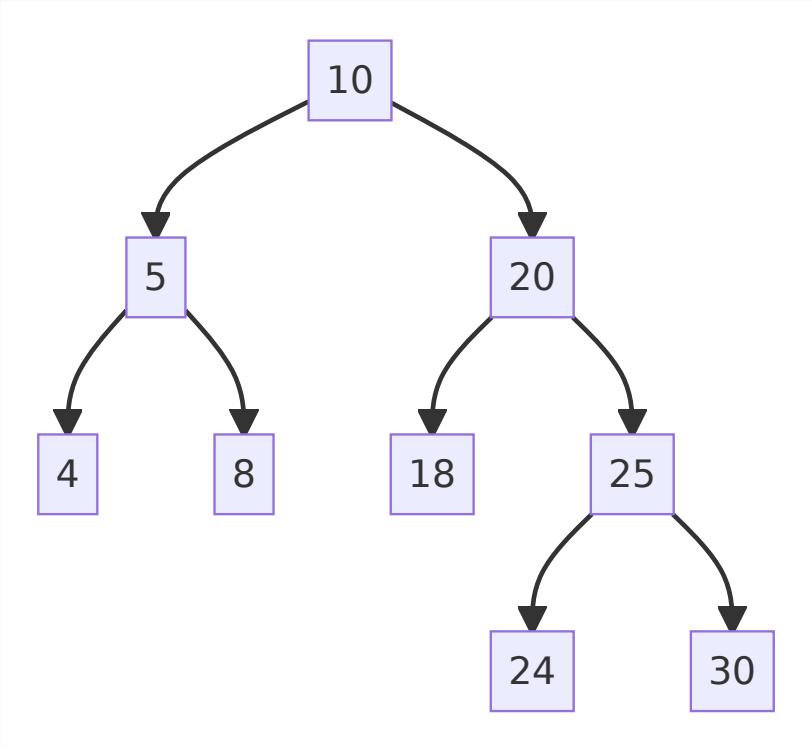


- Si on enlève 10 il se passe quoi?
- On peut pas juste enlever 10 et recoller...
- Proposez une solution bon sang!

La suppression de clé

Cas compliqué

- Le nœud à supprimer à (au moins) deux descendants (10).



- Si on enlève 10 il se passe quoi?
- On peut pas juste enlever 10 et recoller...
- Proposez une solution bon sang!

Solution

- Échange de la valeur à droite dans le sous-arbre de gauche ou ...
- de la valeur de gauche dans le sous-arbre de droite!
- Puis, on retire le nœud.