LYCÉE LECONTE DE LISLE

Arbres et tas binomiaux

Vincent Picard

1
Arbres binomiaux

Arbres binomiaux

Attention à ne pas confondre avec les arbres binaires.

Un **arbre binomial** d'ordre k est :

- pour $k = 0$: une feuille (ou nœud externe)
- pour $k > 0$: un nœud possédant k fils qui sont des arbres binomiaux d'ordres $k-1, k-2, \ldots, 0$ (dans cet ordre)

Arbres binomiaux : questions

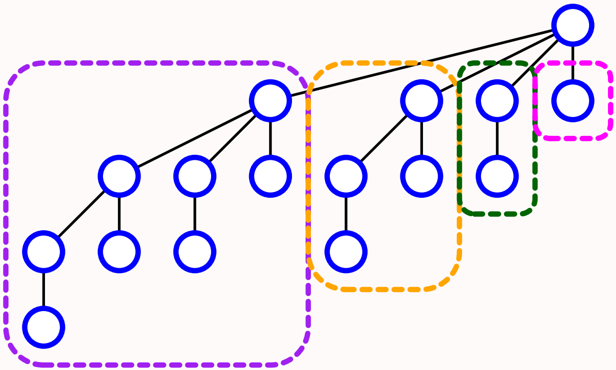
1. Dessiner les arbres binomiaux d'ordres 1, 2, 3, 4.

2. Montrer qu'un arbre binomial d'ordre k est de hauteur k et possède 2^k nœuds.

3. Montrer qu'un arbre binomial d'ordre k possède $\binom{k}{p}$ nœuds à profondeur p. **Indication :** remarquer qu'un arbre binomial d'ordre k s'obtient en ajoutant à un arbre binomial d'ordre $k-1$ un fils gauche qui est aussi un arbre binomial d'ordre $k-1$.

Cette dernière propriété justifie le nom d'arbre binomial.

Un arbre binomial



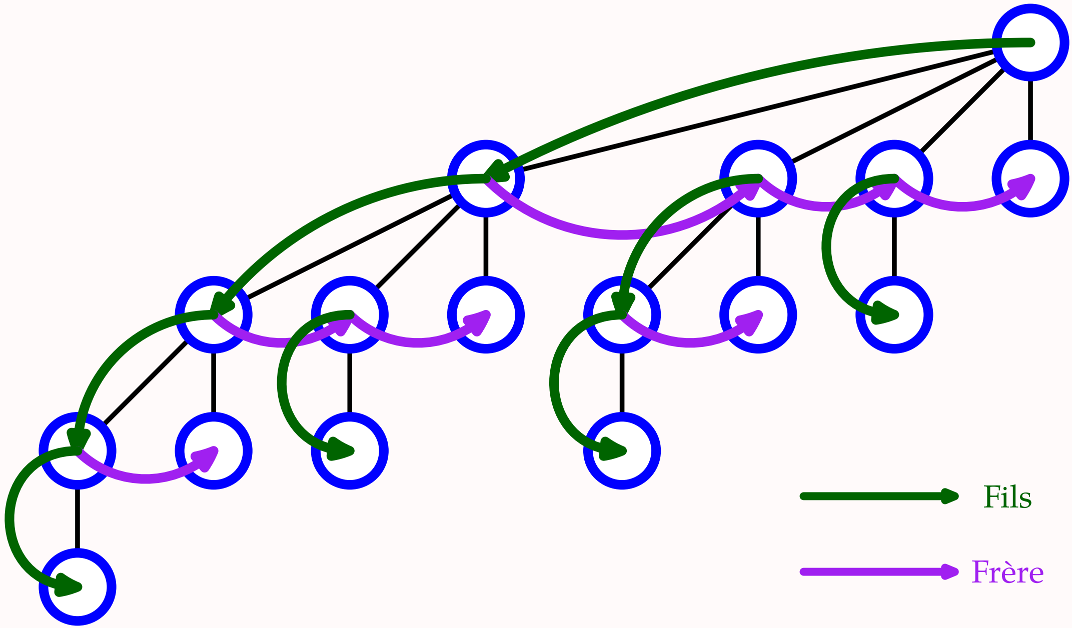
Un arbre binomial d'ordre 4 ($n = 16$)

Arbre binomiaux : implémentation en C

On code les nœuds d'un arbre binomial avec la structure :

```
struct noeud_s {
    int val; /* Etiquette du noeud */
    int ordre; /* Ordre du sous-arbre */
    /* Lien vers son fils de plus grand ordre */
    struct noeud_s *fils; // NULL si aucun fils
    /* Lien vers son petit frère (ordre juste en dessous) */
    struct noeud_s *frere; // NULL si pas de frère
};
typedef struct noeud_s Noeud;
```

Un arbre binomial en C : avec les pointeurs



Arbres binomiaux : exercices en C

Écrire les fonctions C suivantes :

1. `int hauteur(Noeud *arbre)` : calcule la hauteur d'un arbre binomial (sans utiliser son ordre)

2. `int taille(Noeud *arbre)` : calcule la taille d'un arbre binomial (sans utiliser son ordre, ni la question précédente)

3. `bool est_tasmin(Noeud *arbre)` : vérifie si un arbre binomial vérifie la propriété de tas min (tout parent possède une étiquette inférieure ou égale à ses enfants)

Arbres binomiaux : fusion

Les arbres binomiaux possèdent la bonne propriété de pouvoir être facilement fusionnés :

Soit A_1 et A_2 deux arbres binomiaux d'ordre k.

1. On ajoute A_2 comme premier fils (le plus à gauche) de A_1, quel est le résultat ?
2. Implémenter en C la fonction :

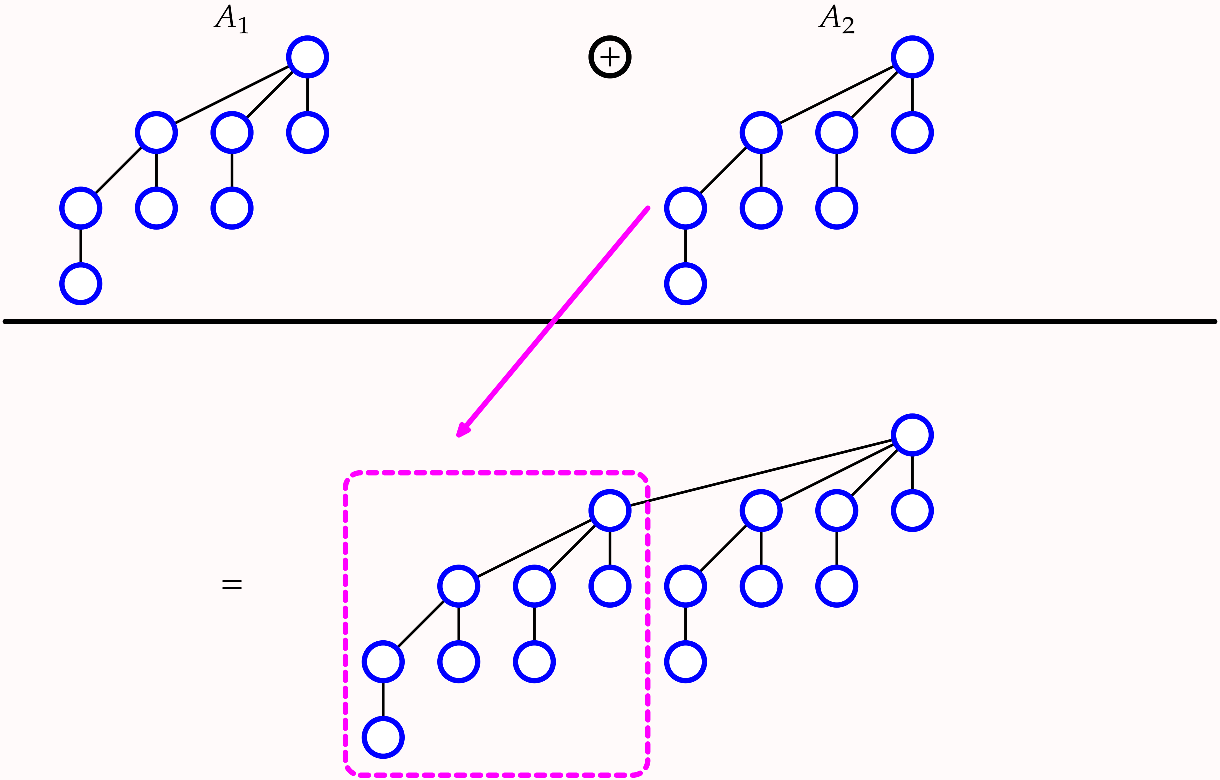
   ```c
   void fusion(Noeud *a1, Noeud *a2)
   ```

   qui réalise cette opération en modifiant seulement les pointeurs dans les deux arbres
3. Quelle est sa complexité ?
4. Implémenter la fonction

   ```c
   Noeud* fusion_tas(Noeud *a1, Noeud *a2)
   ```

   qui réalise la même chose que `fusion` mais qui suppose que `a1` et `a2` vérifient la propriété de tas min. Le résultat doit aussi vérifier la propriété de tas min, on retourne la racine.

La fusion illustrée



2
Tas binomiaux

Tas binomial

Attention à ne pas confondre avec les tas binaires.

Un **tas binomial** est une structure de données implémentant une **file de priorité min** à l'aide d'une forêt d'arbres binomiaux vérifiant :

i. Chaque arbre vérifie la propriété de **tas min**

ii. Pour tout $k \in \mathbb{N}$, il y a au plus un arbre binomial d'ordre k

Tas binomial : questions

1. Dessiner un tas binomial contenant 10 valeurs.

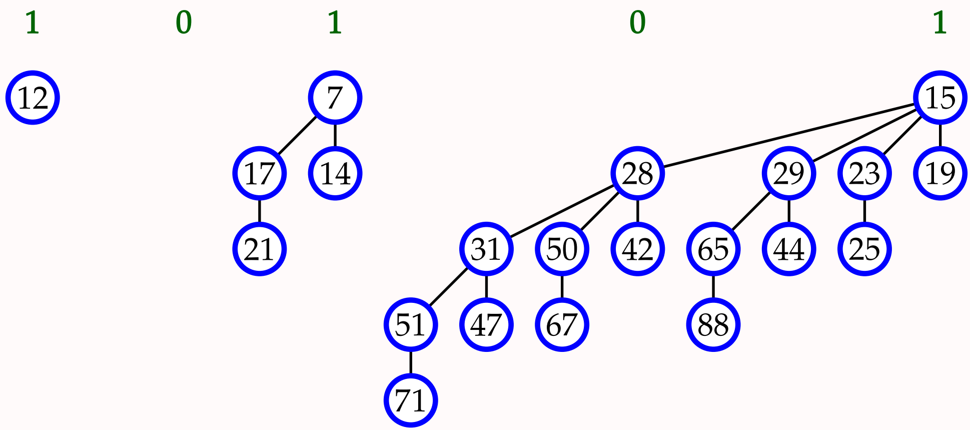
2. Justifier qu'un tas binomial de n valeurs possède une géométrie unique : l'ensemble d'arbres binomiaux utilisés est unique (sans considérer les valeurs sur les nœuds). **[Indication]** : utiliser la représentation en base 2 de n.

3. Ecrire une fonction en C ou en OCaml, prenant en entrée un entier $n \in \mathbb{N}$ et déterminant le nombre d'arbres binomiaux d'un tas binomial de taille n.

4. Montrer que le nombre d'arbres binomiaux d'un tas binomial de taille n est au plus $1 + \log_2(n)$

5. En admettant qu'on possède un accès en $O(1)$ à chacune des racines des arbres, déterminer la complexité pire cas de l'opération de détermination de la valeur minimale dans un tas binomial de taille n.

Tas binomial : un exemple



Un tas binomial contenant $n = 21 = (10101)_2$ valeurs

Représentation en langage C

- Pour simplifier, on représente un tas binomial par un **tableau d'arbres binomiaux**.

```c
typedef Noeud** Tasbinomial;
```

- La case `tas[k]` contient :
  - Soit l'adresse de la racine du seul arbre d'ordre k.
  - Soit `NULL` si l'arbre d'ordre k n'est pas présent.
- On considère que la taille du tableau est $N = 32$. Les arbres seront d'ordre 31 au maximum, ce qui permet d'avoir au plus $2^{32} - 1 \simeq 4$ milliards de valeurs.

```c
#define N 32
```

Création d'un tas binomial vide

Pour créer un tas binomial vide, on alloue sur le tas un tableau de taille N où toutes les cases sont initialisées à NULL (pas d'arbre).

```
Tasbinomial creer_tasbinomial() {
    Tasbinomial t = malloc(N * sizeof(Noeud*));
    for (int i = 0; i < N; i++) {
        t[i] = NULL;
    }
    return t;
}
```

Valeur minimale

- Écrire une fonction de signature

```c
int valeur_min(Tasbinomial t)
```

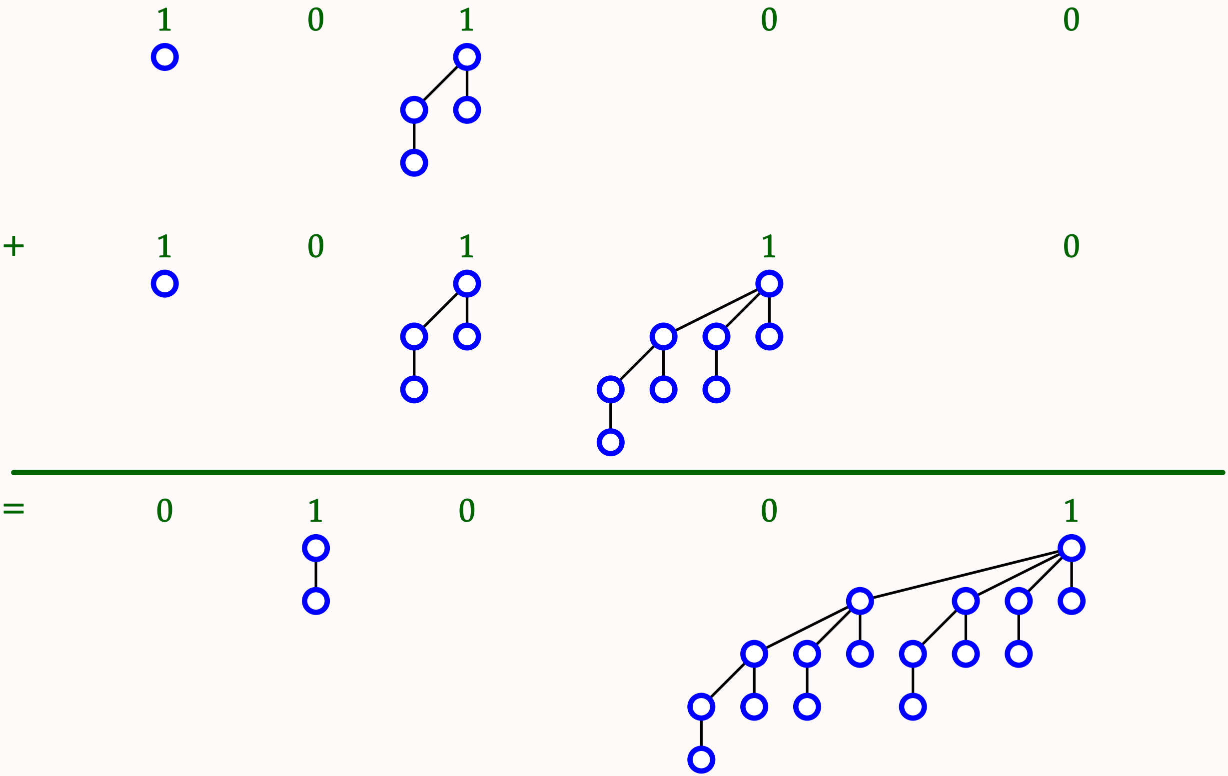
  renvoyant la valeur minimale d'un tas binomial non vide.

- **Indication :** on pourra utiliser la valeur `INT_MAX` du plus grand `int` représentable en machine. Cette valeur est définie dans `limit.h`

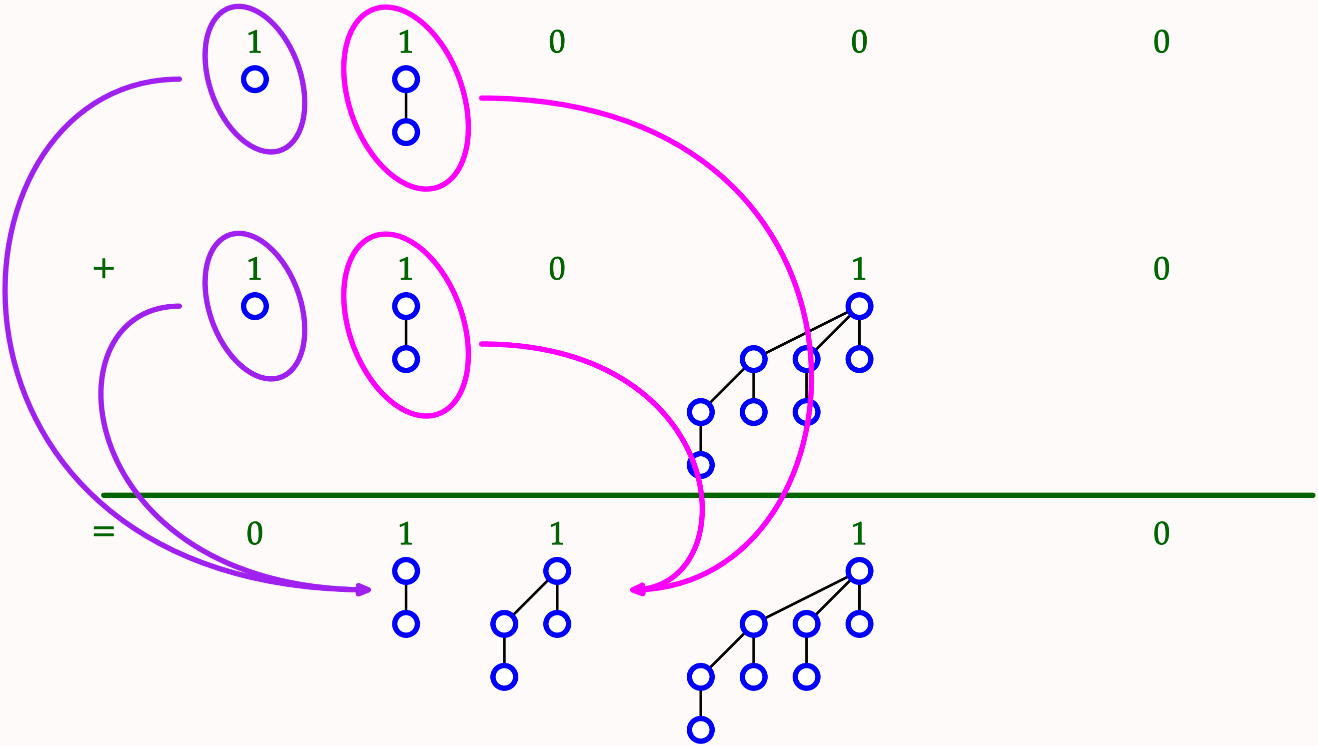
Fusion de deux tas binomiaux

- L'avantage d'utiliser des tas binomiaux plutôt que des tas binaires est la possibilité de **fusionner** facilement deux tas binomiaux en un seul.
- Lorsqu'on fusionne deux tas binomiaux on réunit les arbres des deux forêts.
- Il se peut qu'on obtienne ainsi deux arbres d'ordre k : on applique alors la fusion de ces deux arbres binomiaux pour obtenir un arbre d'ordre $k + 1$
- En répétant les fusions, on élimine ainsi tous les doublons d'arbres de même ordre.
- Cet algorithme est très similaire à l'**addition en base 2**

Fusion de deux tas binomiaux : exemple 1



Fusion de deux tas binomiaux : exemple 2



Fusion : programmation

- Programmer une fonction

  ```
  Tasbinomial fusion_tasbinomial(Tasbinomial t1, Tasbinomial t2)
  ```

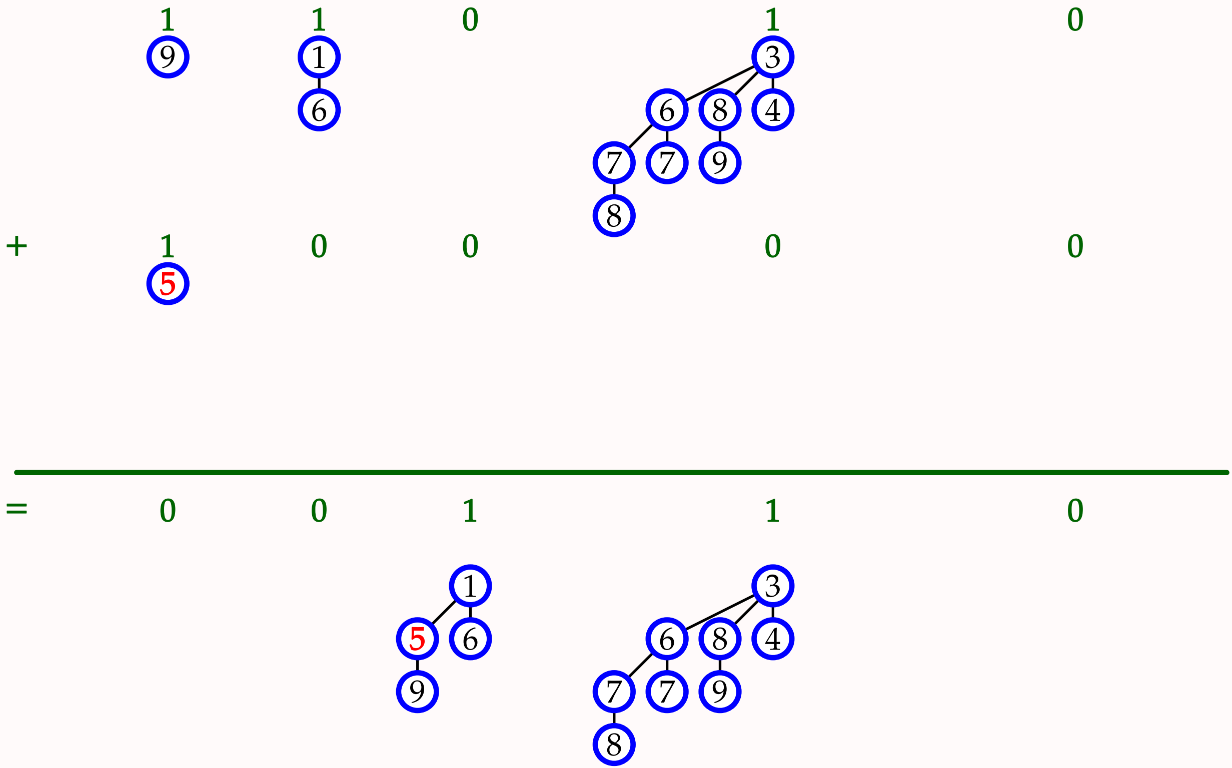
  implémentant la fusion de deux tas binomiaux.
- Cette fonction devra libérer la mémoire allouée pour les tableaux `t1` et `t2`.
- La fonction renvoie un nouveau tableau alloué sur le tas.
- Les nœuds de `t1` et `t2` ne sont modifiés qu'au travers de `fusion_tasmin` : aucun nœud n'est créé ni détruit.

Insertion d'une valeur

Pour insérer une valeur x dans un tas binomial t_1 on effectue les étapes suivantes:

- Créer un nouvau tas binomial vide t_2.
- Dans t_2 : créer un unique arbre d'ordre 0 contenanant la valeur x.
- Fusionner t_1 et t_2.

Insertion d'une valeur : exemple

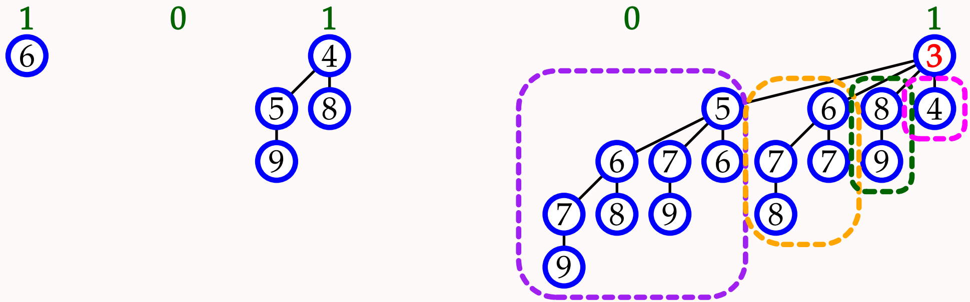


Extraction de la valeur minimale

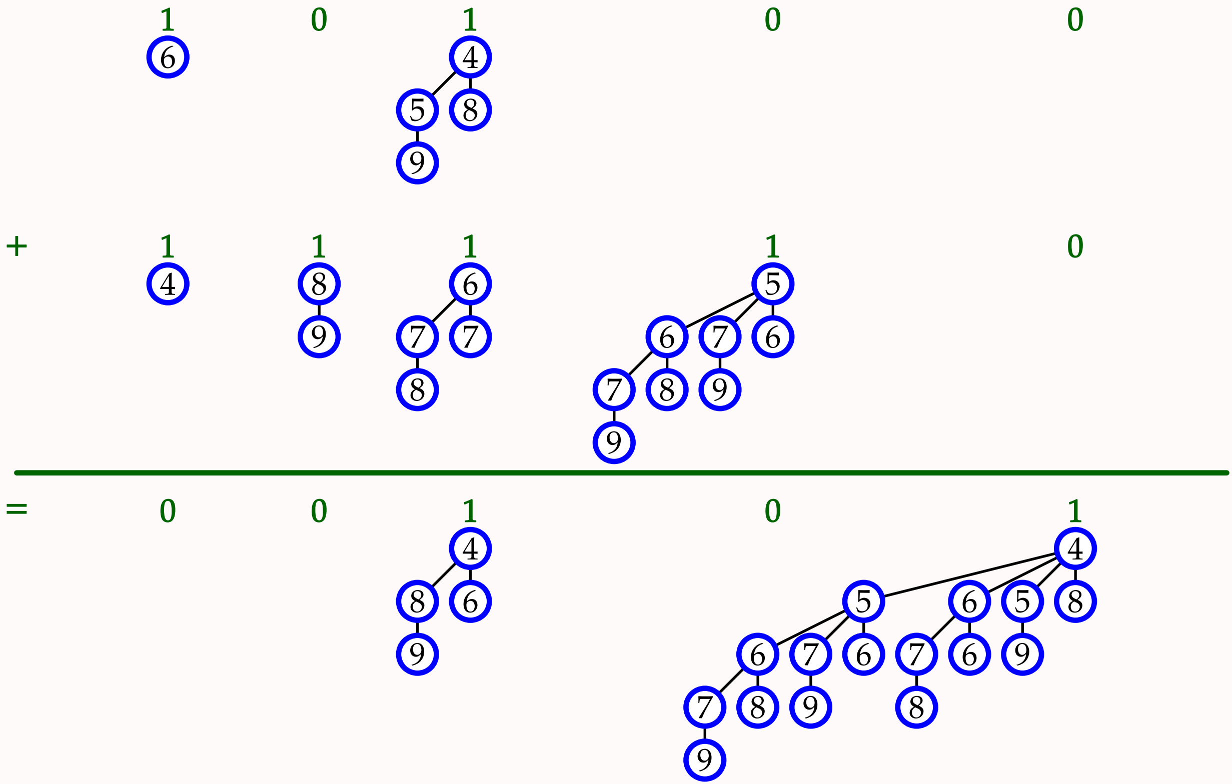
Pour supprimer la valeur minimale d'un tas binomial t_1 **non vide** on effectue les étapes suivantes:

- Repérer la racine r_{min} contenant la valeur minimale.
- Si r_{min} est l'arbre d'ordre 0, on supprime simplement cet arbre.
- Sinon on créé un tas binomial t_2 à partir des fils de r_{min}.
- Puis on supprime r_{min}.
- Enfin on fusionne t_1 et t_2.

Extraction de la valeur minimale : suppression



Extraction de la valeur minimale : fusion



Utilisation : exemple complet

```
int main() {
    Tasbinomial t = creer_tasbinomial();
    Tasbinomial t2 = creer_tasbinomial();
    t = inserer(t, 18);
    t = inserer(t, 42);
    t = inserer(t, 98);
    t = inserer(t, 50);
    t = inserer(t, 70);
    t2 = inserer(t2, 44);
    t2 = inserer(t2, 71);
    t2 = inserer(t2, 6);
    t2 = inserer(t2, 31);
    t2 = inserer(t2, 30);
    Tasbinomial t3 = fusion_tasbinomial(t, t2);
    for (int k = 0; k < 12; k++) {
        printf("valeur min : %d\n", valmin(t3));
        t3 = supprimer(t3);
    }
}
```

Complexité pire cas des opérations

Notons n le nombre de valeurs dans la file de priorité.

Opération	Tas binaire	Tas binomial
Lire la valeur minimale	$O(\log_2(n))$	$O(\log_2(n))$
Insérer une valeur	$O(\log_2(n))$	$O(\log_2(n))$
Extraire la valeur minimale	$O(\log_2(n))$	$O(\log_2(n))$
Diminuer[1] une valeur	$O(\log_2(n))$	$O(\log_2(n))$
Fusionner deux files	✗	$O(\log_2(n))$

[1] Dans les deux structures, il suffit de percoler vers le haut

Améliorer l'implémentation

- On peut améliorer notre implémentation en codant un tas binomial comme une **liste chaînée** d'arbres binomiaux d'ordres croissants.
- Cela évite de stocker et pacourir les arbres `NULL`.
- Il faut alors coder la **fusion** de deux tas binomiaux en parcourant intelligemment les deux listes simultanément et en produisant une nouvelle liste en sortie.
- Lors de la **suppression**, il faut renverser la liste des fils car les fils sont triés par ordre décroissant d'ordre.

3
Solutions

Hauteur d'un arbre binomial

On remarque que la branche la plus profonde est située le plus à gauche, donc il suffit de descendre au maximum sur le fils de plus grand ordre :

```
int hauteur(Noeud *arbre) {
    if (arbre == NULL) {return -1;} // par convention
    int h = 0;
    Noeud *actuel = arbre;
    while (actuel->fils != NULL) {
        h += 1;
        actuel = actuel->fils;
    }
    return h;
}
```

Taille d'un arbre binomial

On panache le style impératif (pour parcourir les fils) avec le style fonctionnel (pour calculer récursivement la taille de chaque fils) :

```c
int taille(Noeud *arbre) {
    if (arbre == NULL) {return 0;}
    int t = 1;
    Noeud *actuel = arbre->fils;
    while (actuel != NULL) {
        t += taille(actuel);
        actuel = actuel->frere;
    }
    return t;
}
```

Vérification de la propriété de tas min

De même on allie un parcours itératif des fils avec une vérification récusrive de la propriété sur chaque sous-arbre :

```
bool est_tasmin(Noeud *arbre) {
    assert(arbre != NULL);
    Noeud *actuel = arbre->fils;
    while (actuel != NULL) {
        if (!est_tasmin(actuel) || actuel->val > arbre->val) {
            return false;
        }
        actuel = actuel->frere;
    }
    return true;
}
```

Fusion de deux arbres binomiaux

Cette fonction fusionne les arbres binomiaux a_1 et a_2 de même ordre, en plaçant a_2 comme fils de a_1 de plus grand ordre.

```c
void fusion(Noeud *a1, Noeud *a2) {
    assert(a1 != NULL && a2 != NULL);
    assert(a1->ordre == a2->ordre);
    a2->frere = a1->fils;
    a1->fils = a2;
    a1->ordre += 1;
}
```

La complexité est en $O(1)$.

Fusion de deux arbres binomiaux en conservant la propriété de tas min

On compare les racines de a_1 et a_2 pour savoir qui sera la racine du résultat, puis on utilise la fonction précédente dans le bon sens :

```
Noeud* fusion_tasmin(Noeud *a1, Noeud *a2) {
    assert(a1 != NULL && a2 != NULL);
    assert(a1->ordre == a2->ordre);
    if (a1->val <= a2->val) {
        fusion(a1, a2);
        return a1;
    } else {
        fusion(a2, a1);
        return a2;
    }
}
```

Fusion de deux tas binomiaux

```
Tasbinomial fusion_tasbinomial(Tasbinomial t1, Tasbinomial t2) {
    Noeud* carry = NULL; // Retenue
    Tasbinomial r = malloc(N * sizeof(Noeud*));
    for (int i = 0; i < N; i++) {
        // On calcule t1[i] + t2[i] + retenue
        if (carry == NULL) {
            if (t1[i] == NULL) {
                r[i] = t2[i];
            } else if (t2[i] == NULL) {
                r[i] = t1[i];
            } else {
                r[i] = NULL;
                carry = fusion_tasmin(t1[i], t2[i]);
            }
        } else {
            if (t1[i] == NULL && t2[i] == NULL) {
                r[i] = carry;
                carry = NULL;
            } else if (t1[i] == NULL) {
```

```c
            r[i] = NULL;
            carry = fusion_tasmin(carry, t2[i]);
        } else if (t2[i] == NULL) {
            r[i] = NULL;
            carry = fusion_tasmin(carry, t1[i]);
        } else {
            r[i] = carry;
            carry = fusion_tasmin(t1[i], t2[i]);
        }
    }
}
// Vérification du dépassement de capacité
assert(carry == NULL);
free(t1);
free(t2);
return r;
}
```

Insertion

```
/* Insere une nouvelle valeur, la mémoire de t est libérée */
Tasbinomial inserer(Tasbinomial t, int val) {
    Tasbinomial t2 = creer_tasbinomial();
    t2[0] = malloc(sizeof(Noeud));
    t2[0]->ordre = 0;
    t2[0]->val = val;
    t2[0]->fils = NULL;
    t2[0]->frere = NULL;
    Tasbinomial r = fusion_tasbinomial(t, t2);
    return r;
}
```

Suppression de la valeur minimale

```
/* Supprime la valeur minimale, la mémoire de t est libérée */
Tasbinomial supprimer(Tasbinomial t) {
    /* On repere l'arbre de racine minimale */
    int min = INT_MAX;
    int imin = -1;
    for (int i = 0; i < N; i++) {
        if (t[i] != NULL && t[i]->val < min) {
            min = t[i]->val;
            imin = i;
        }
    }
    assert(imin >= 0);
    if (imin == 0) { /* Cas ou le min est dans l'arbre-feuille */
        free(t[0]);
        t[0] = NULL;
        return t;
    }

    /* On créé un nouveau tas binomial à partir des fils de la racine
```

```
    Tasbinomial t2 = malloc(N * sizeof(Noeud*));
    for (int i = 0; i < N; i++) {
        t2[i] = NULL;
    }
    Noeud* actuel = t[imin]->fils;
    for (int j = imin-1; j >= 0; j--) { // Les fils d'ordre i-
1 à 0 forment t2
        t2[j] = actuel;
        actuel = actuel->frere;
    }
    free(t[imin]); // On libre le noeud racine
    t[imin] = NULL;

    Tasbinomial r = fusion_tasbinomial(t, t2);
    return r;
}
```

4
Et avec OCaml ?

Types

```
type 'a arbre = {
    valeur: 'a;
    ordre : int;
    fils : 'a arbre list (* par ordre décroissant *)
};;

type 'a tas_binomial = 'a arbre list;;

let tas_vide = [];;
```

Exercices

Implémentez les fonctions suivantes :

1. `hauteur_arbre : 'a arbre -> int`
2. `taille_arbre : 'a arbre -> int`
3. `fusion_arbre : 'a arbre -> 'a arbre -> 'a arbre` (en conservant la propriété de tas)
4. `val_min : 'a tas_binomial -> 'a`
5. `fusion_tasbinomial : 'a tas_binomial -> 'a tas_binomial -> 'a tas_binomial`
6. `inserer : 'a -> 'a tas_binomial -> 'a tas_binomial`
7. `supprimer_min : 'a tas_binomial -> 'a tas_binomial`