

SÉANCE 6 | TÉLÉCODE

ARBRES D'INTERVALLES



PLAN

01

PRÉSENTATION

02

ARBRE D'INTERVALLE POUR LA SOMME

03

IMPLÉMENTATION

04

AUTRES UTILISATIONS

05

PROBLÈMES

1. PRÉSENTATION

- Objectif :
 - Répondre à des requêtes sur un intervalle (ex : somme $a[i\dots j]$, maximum sur $[i\dots j]$)
 - Faire des modifications sur une valeur ou sur un intervalle (ex : fixer $a[i] = x$, fixer $a[i\dots j] = x$, faire $a[i\dots j] += x$)
- Très flexible en termes de possibilités



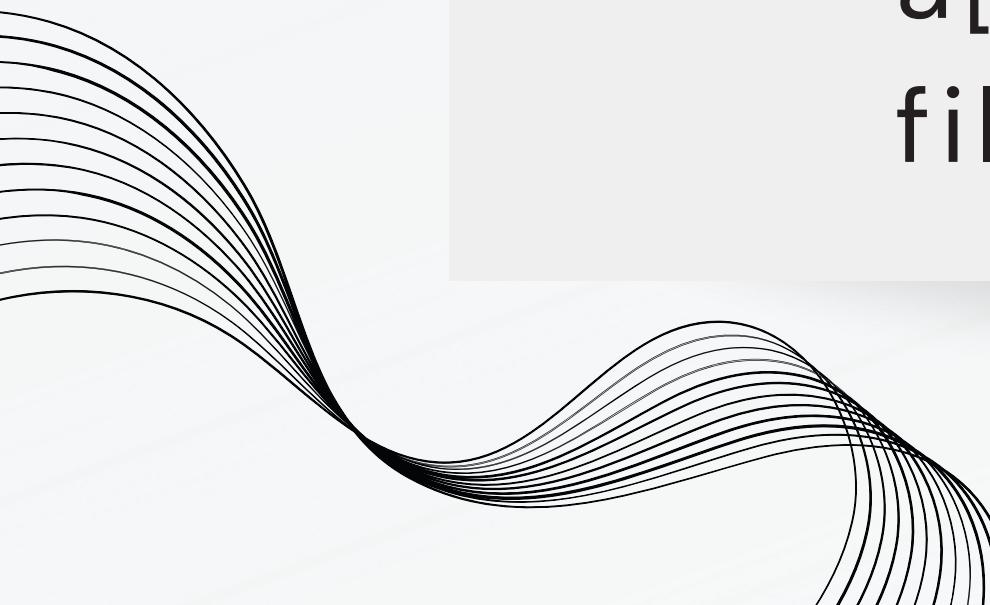
1. PRÉSENTATION

- Naïvement :
 - Complexité $O(n)$ pour requête et modification
 - Si “r” requêtes => $O(r * n)$
- Arbres d’intervalles (arbre binaires):
 - $\log(n)$ à chaque requête
 - $O(r * \log n)$
 - $O(n)$ en mémoire seulement !



2. ARBRE D'INTERVALLE POUR LA SOMME

- Idée : arbre binaire
 - Feuille = requête d'un élément seulement
 - On **combine** les nœuds en remontant (ex : parent des nœuds avec somme $a[i...j]$ et $a[j+1...k]$ contiendra la valeur $a[i...k]$ qui est juste la somme des 2 fils)



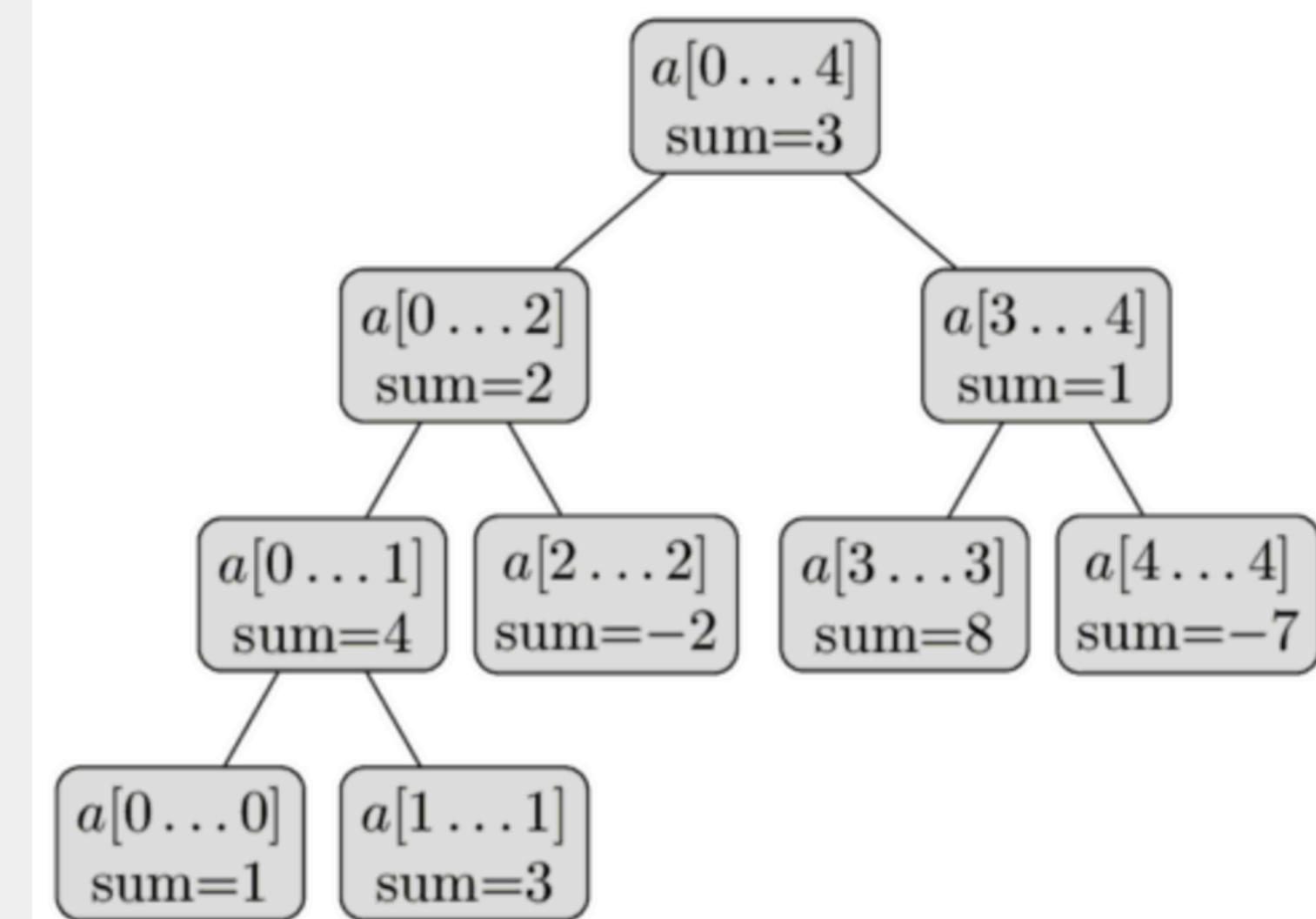
2. ARBRE D'INTERVALLE POUR LA SOMME

- Détails d'implémentation :
 - Information stockée dans chaque nœud (ici somme de l'intervalle)
 - Fonction merge (réunion de 2 nœuds)
- Important :
 - Fils du nœud i : $2i$ (gauche), $2i+1$ (droit)
 - Parent du nœud i : $i/2$



2. ARBRE D'INTERVALLE POUR LA SOMME

- $a = [1, 3, -2, 8, -7]$



3. IMPLÉMENTATION

Construire l'arbre de requêtes à partir d'une liste $a[0..n-1]$
-> à appeler avec **build(a, 1, 0, n-1)**

```
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

3. IMPLÉMENTATION

Fonction qui répond à une requête
à appeler avec **sum(1, 0, n-1, l, r)**

```
int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
        + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}
```

3. IMPLÉMENTATION

Fonction de mise à jour d'un élément
-> on veut avoir **a[pos] = new_val**

```
void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

4. AUTRES UTILISATIONS

- Max/ min
- Max et nombre d'apparitions
- PGCD/ PPCM
- Plus petit indice i tel que somme jusqu'à i soit plus grande que x
- [...]
- Pour mises à jour de segments $[i...j] \rightarrow$ regarder **lazy propagation**



4. AUTRES UTILISATIONS

- Maximal subsegment sum (idée très puissante)

```
struct data {
    int sum, pref, suff, ans;
};

data combine(data l, data r) {
    data res;
    res.sum = l.sum + r.sum;
    res.pref = max(l.pref, l.sum + r.pref);
    res.suff = max(r.suff, r.sum + l.suff);
    res.ans = max(max(l.ans, r.ans), l.suff + r.pref);
    return res;
}
```

5. PROBLÈMES

CSES (range queries) :

Dynamic range minimum queries
(Prefix sum queries)

Codeforces :

2072/E

2074/D