921

Algorithmes de recherche et structure de données associées.

Introduction :

* Motivation : Stocker et exploiter des données est une utilisation des plus communes de l'informatique. Savoir retrouver une donnée précise dans une collection est un problème récurrent.

* Problème de recherche :
  - entrée : K une collection d'objet ; $E \subseteq K$ et $k \in K$.
  - sortie : $k \in K$.

I. Recherche dans un dictionnaire.

Objectif : Recherche d'un élément quand $K = \{(clé, valeur)\}$ et E est un dictionnaire (structure abstraite).

A. L'opération recherche suit une loi uniforme sur les clés.

Objectif : Recherche d'un élément quand $K = \{(clé, valeur)\}$ et E est un dictionnaire. De plus, k est uniformément distribué sur E.

Algorithme 1 : Recherche naïve : on compare k à tous les éléments de E.
→ Structures de données associées : liste, tableau
→ Complexité temporelle dans le pire cas : $O(n)$.

Algorithme 2 : Recherche dichotomique. [1, p. 179]
→ Structure de données associée : tableau trié

```
dicho(k, E, g, d, res) :        | g,d ∈ {0,..., |E|-1}
                                | res ∈ {0,..., |E|}
  Si g ≤ d alors
     m := (g+d) div 2
     Si k = E[m] alors res := m
     Sinon Si k < E[m]
           alors dicho(k, E, g, m-1, res)
           Sinon dicho(k, E, m+1, d, res)
  Sinon res := 0
```

→ Complexité temporelle dans le pire cas : $O(\log n)$

Algorithme 3 : Recherche supportée par un ABR équilibré : recherche dans un ABR.

Définition 4 : [1, p. 197] Un arbre binaire de recherche (ABR) est un arbre binaire étiqueté tel que pour tout noeud x d'étiquette n de l'arbre :
* les éléments de tous les noeuds du fils gauche de x sont inférieures à n ;
* les éléments de tous les noeuds du fils droits de x sont supérieures à n.

(Figures en annexe : 1 et 2)

Définition 5 : [1, p. 221] Un ABR à n noeuds est dit équilibré si sa hauteur est en $O(\log n)$.

→ Implémentations : AVL [1, p. 224]
arbre rouge - noir [2, p. 287] (Figure en annexe : 3)
→ Complexité temporelle dans le pire cas : $O(\log n)$

Application 6 : Recherche dans le cadre de la géométrie algorithmique.

→ Problème : L'intersection de n droites par la méthode de balayage. [2, p. 940]
On cherche un élément dans K est l'ensemble des points du plan et E l'ensemble des points d'intersection de deux droites.
. Implémentation : arbre rouge noir
. Complexité temporelle dans le pire cas : $O(n \log n)$.

→ Problème : Recherche d'un point dans un nuage. [3, p. 309]
On cherche un élément dans $K = \{(clé, valeur)\}$ avec clé dans l'ensemble des points du plan et E est l'ensemble des points du nuage.
. Solution : on utilise les quad-tree
(Figure en annexe : 4)

B – La recherche n'est pas uniforme sur l'ensemble des clés.

Objectif : Recherche d'un élément quand $K = \{(\text{clé}, \text{valeur})\}$ et E est un dictionnaire et k suit une distribution sur E.

Algorithme 7 : Recherche auto-adaptative [1, p. 175]
- → On met en première position l'élément que l'on vient de rechercher.
- → Structure de données associée : liste
- → Complexité temporelle asymptotique en le nombre de recherche et en moyenne : $O\left(\frac{m}{\log m}\right)$ [4]

Algorithme 8 : Recherche supportée par un ABR optimal : recherche dans un ABR. [2, p. 368]

Définition 9 : Un ABR optimal minimise la complexité espérée de la recherche en fonction de la distribution sur les clés.

(Figure en annexe : 5)

→ Implémentation : arbre splay.

C – Les clés ne possèdent pas nécessairement un ordre total. [2, p. 238]

Objectif : Recherche d'un élément quand $K = \{(\text{clé}, \text{valeur})\}$ où les clés sont à valeur dans un ensemble qui n'admet pas nécessairement un ordre total.

Principe 10 : Pour chaque valeur de clé, une fonction de hachage h lui associe un entier représentant un indice de tableau qui correspond à la case qui va la stocker.

Définition 11 : Deux éléments $u \neq v$ sont en collision ssi $h(u) = h(v)$

→ Implémentation : table de hachage.
- Gestion des collisions par chaînage simple.
- Complexité temporelle dans le pire cas : $O(n)$

[2, p. 258]

Définition 12 : On parle de hachage parfait si la recherche s'effectue en $O(1)$ (Figures en annexe 6)

D. Le dictionnaire doit être stocké sur un disque externe

Objectif : Recherche d'un élément quand $K = \{(\text{clé}, \text{valeur})\}$ et E est un dictionnaire qui ne peut pas être stocké en mémoire vive.

[2, p. 447]

Définition 13 : Un arbre B est un arbre de recherche qui minimise le nombre d'accès au disque externe. **DEV**

Remarque 14 : Un arbre B peut être étendu en un arbre B^+ pour faciliter les recherches séquentielles.

Application 15 : Base de données relationnelle.
(Figures en annexe 7 ; 8)

III – Recherche dans un texte.

Objectif : Recherche d'un élément lorsque K est l'ensemble des mots de Σ^* (Σ alphabet) et E l'ensemble des facteurs d'un texte.

Applications 16 : Recherche d'un mot dans un fichier texte, d'une chaîne de caractère dans une séquence ADN, …
→ Structure de données associée : tableau de la taille de la chaîne.

Algorithme 17 : Recherche naïve. [2, 908]

entrée : T. texte P. motif sortie : décalage

```
Recherche-naïve (T, P)
  n := T.longueur ; m := P.longueur
  pour s = 0 à n - m
    si P[1, ..., m] = T[s+1, ..., s+m]
    alors retourner s
```

→ Complexité temporelle dans le pire cas : $O((n-m+1)m)$

A – Algorithme de Rabin-Karp [2, 910]

entrée : T texte P motif sortie : décalage

Hypothèse : On considère que $\Sigma = \{0, \dots, 9\}$.

Algorithme 18 : Algorithme de Rabin-Karp

```
Rabin-Karp (T, P)
  n := T.longueur ; m := P.longueur ; h := d^(m-1) mod 10
  p := 0 ; t_0 := 0
  pour i = 1 à m
    p := (dp + P[i]) mod q ; t_0 := (dt_0 + T[i]) mod q
  pour s = 0 à n - m
    si p = t_s alors
      si P[1 ... m] = T[s+1 ... s+m] alors retourner s
    t_{s+1} = (d (t_s - T[s+1] h) + T[s+m+1]) mod q
```

→ Complexité temporelle du prétraitement : $\Theta(m)$
→ Complexité temporelle dans le pire cas : $O((n-m+1)m)$.
(Figures en annexe 9)

B - Algorithme de Knuth-Morris-Pratt [2, p. 915]

Hypothèse : Le motif est fixe et connu à l'avance.

Définition 19 : La fonction suffixe associée au motif P, $\sigma : \Sigma^* \to \{0, \dots, m\}$ donne la longueur du plus long préfixe de P qui est suffixe de $x \in \Sigma^*$.

Définition 20 : On définit l'automate des occurences associé à $P[1, \dots m]$ comme suit : $Q = \{1, \dots, m\}$; $q_0 = 0$; $q_F = m$; $\delta(q, a) = \sigma(P_q a)$ pour $a \in \Sigma$, $q \in Q$ et $P_q = P[1, \dots q]$

Algorithme 21 : Calcul de δ.

```
Calcul-δ (P, Σ)
  m := P.longueur
  pour q = 0 à m
    pour a ∈ Σ
      k := min (m+1, q+2)
      repéter
        k := k-1
      jusqu'à P_k suffixe de P_q a
      δ(q, a) := k
  retourne δ
```

entrée : P motif
Σ alphabet
sortie : δ

→ Complexité temporelle du prétraitement : $O(m|\Sigma|)$

Algorithme 22 : Recherche dans l'automate

```
Recherche-automate (T, δ, m)
  n := T.longueur ; q := 0
  pour i = 1 à n
    q := δ(q, T[i])
    si q = m
      retourne i-m.
```

entrée : T texte
δ : fonction de transition
m longueur du motif
sortie : décalage

→ Complexité temporelle dans le pire cas : $\Theta(n)$

Définition 23 : La fonction préfixe associée au motif P, $\pi : \{1, \dots, m\} \mapsto \{0, \dots, m-1\}$ donne la longueur du plus long préfixe de P qui est suffixe de P_q où $q \in \{1, \dots, m\}$.

Algorithme 24 : Algorithme de Knuth-Moris-Pratt
→ Idée : construction intelligente de l'automate précédent pour que la fonction δ puisse être calculée à la volée.
→ Complexité temporelle du prétraitement dans le pire cas : $\Theta(m)$
→ Complexité temporelle dans le pire cas : $\Theta(n)$

DEV

IV - Ouverture

Problème 25 : Problème de la recherche du min et du max : On cherche un élément, k, dans K un ensemble ordonné et $E \subseteq K$, $k = \inf_K E$ ou $k = \sup_K E$

Problème 26 : Union-find et partition d'un ensemble, On cherche un élément, k, dans K l'ensemble des entiers et $E = \{(n, \bar{n})\}$ où $\bar{n}$ représente son représentant de sa classe d'équivalence.

Problème 27 : Problème de la recherche non-associative : on ne recherche pas sur la valeur et mais également sur son contexte : recherche "temporelle" (l'élément qui est depuis le plus longtemps), du k^{ième} élément, dans la structure

Problème 28 : Problème d'accessibilité dans un graphe. On cherche un élément k, dans K l'ensemble des états accessibles de ce d'un graphe et E l'ensemble des états accessibles de ce graphe.

Références :

[1] Froidevaux, Gaudel, Types de données et algorithmes

[2] Rivest, Stein, Cormen, Leiserson, Algorithmique, 3ème édition.

[3] Overmars, Schwarzkopf, Berg, van Kreveld, Computational Geometry

[4] Knuth, The Art of Computer Programming, vol 3

Annexe :

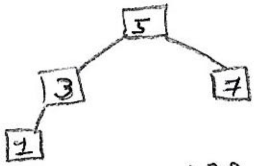


Figure 1: Un ABR équilibré

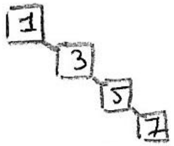


Figure 2: Un ABR déséquilibré

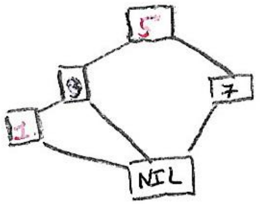


Figure 3: Un arbre rouge-noir

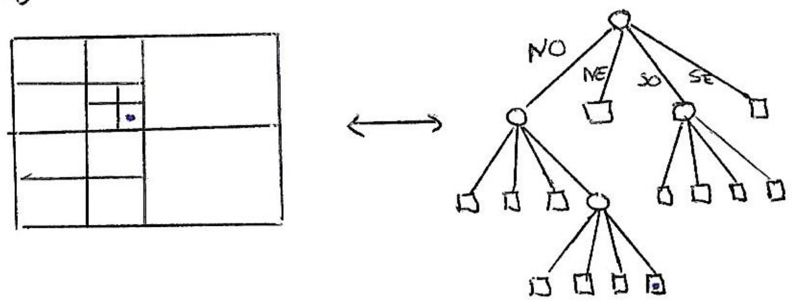
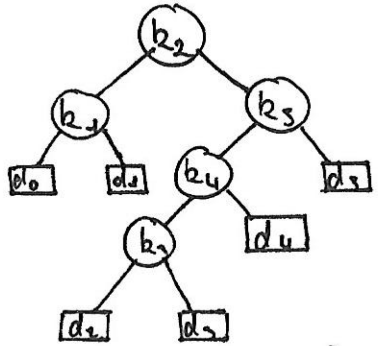


Figure 4: Un exemple de Quad-Tree.



i	0	1	2	3	4	5
p_i		0,15	0,1	0,05	0,1	0,2
q_i	0,05	0,1	0,05	0,05	0,05	0,1

Figure 5: Un exemple d'ABR optimal.

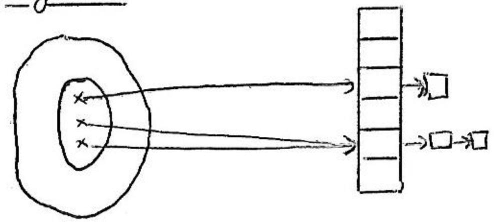


Figure 6: Hachage avec résolution des collision par chainage simple.

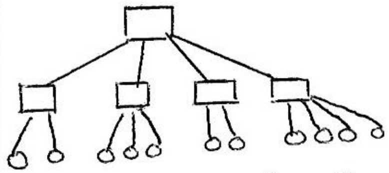


Figure 7: Un arbre B

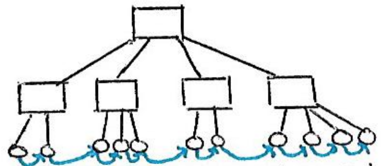


Figure 8: Un arbre B+

On cherche P = 2 1 2

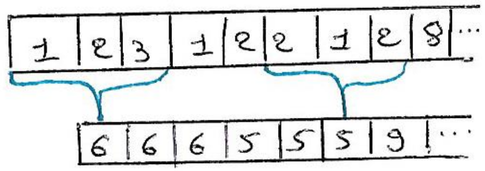


Figure 9: Algorithme de Rabin-Karp.