02/06 2015

Structures de données : exemples et applications.

901

I Type de données

Définition 01 : Type de données abstrait ; Structure de donnée
Un type de données abstrait est une spécification mathématique d'un ensemble de données et des opérations qu'elles peuvent effectuer
Un type de données est l'implantation effective d'un type abstrait

On décrit un type abstrait par :
- sa signature
- des axiomes
- des préconditions

exemple : dictionnaire

II Structures adaptées à la collection d'objets

1. Structures séquentielles

a. Les listes itératives

Sorte : T-ab ; Place

Opérations :
- créer_tableau
- accès_case
- insérer_elem
- contenu_case
- longueur_tableau

b. Les listes chaînées

Sorte : Liste

Opérations :
- créer
- est_vide
- tête
- ajoute_elem
- queue

Implantation :
- contigue : représentation par un tableau
- récursive : $\mathbb{L} = Vide + el \times \mathbb{L}$

c. Piles

Sorte : Pile

opérations :
- créer_pile
- dépile
- est_vide
- empile
- Sommet_pile

Implantation :
- contigüe : un tableau et un entier représentant le sommet de la pile
- chaînée : une liste chaînée

application : Tri par insertion ; fonction annuler

d. Files

Sorte : File

- créer_File
- défiler
- est_vide
- enfiler
- premier_elem

Implantation :
- contigüe : un tableau et 2 entiers représentant le début et la fin de la file.
- double chaînage
- chaînage circulaire

application : gestion d'une imprimante

e. Files de Priorité

Sorte : FileP

Raffine : File

- créer
- insérer
- est_FileP
- est_vide
- défile

Implantation : à l'aide de Tas

application : Algorithme de Dijkstra.

2. Structures arbores-centes

a. Arbre binaire

Sortes: Noeud ; Arbre

Opérations:

- créer
- construire
- racine
- fils_gauche
- fils_droit
- contient

Implantation: • Récursive : ABR = Vide | Noeud (R ; f_g ; f_d)

Application: Représentation d'expressions en logique des propositions.

exemple d'arbres binaires:

définition 02: Tas ; Tas max

Un tas est un arbre binaire dont tous les niveaux sont complètement remplis, sauf éventuellement le dernier et dont les feuilles sont les plus à gauche possible.
Un tas est un tas max si la racine de tout sous arbre est plus grande que tout les éléments du sous-arbre.

PVP

applications: Tri par Tas
L'algorithme du tri par tas permet de trier un tableau de taille n en temps $O(n \ln(n))$.

applications: • Implantation des Files de priorité
• Codage de Huffman : représentation d'un message par un code binaire construit au moyen d'un arbre binaire

b. Graphes

Sortes: Graphe ; Noeud ; arête.

Opérations:

- créer
- ajoute_sommet
- ajoute_arrete
- est_sommet
- est_arrete

Implantation: • Tableaux
• Listes

application: • Algorithmes de G.P.S.
• Ordonnancement

III Des structures de données adaptées à la recherche

1. Tables de Hachage

définition 07: Fonction de Hachage
Soit U, un ensemble de clefs et soit T un tableau de taille m.
Une fonction de hachage est une application $h : U \to [\![1, m]\!]$

Lorsque U est grand, il n'est pas toujours possible de garantir $|U| \leqslant m$. h est alors non-injective et apparait un phénomène : la collision, c'est-à-dire l'existence de 2 clefs u_1 et u_2 distinctes telle que $h(u_1) = h(u_2)$.
Le principe étant de ranger la clé u dans la case h(u), il faut résoudre ceci.

chaînage: Lorsque plusieurs clefs entrent en collision, on stocke les éléments dans une liste chaînée.

application: gestion d'un vestiaire de m portes manteaux et avec $n \gg m$ manteaux.

Type abstrait

• insérer • rechercher • supprimer

Implantation : Représentation de la table par un tableau
Représentation des alvéoles par des listes chaînées.

Théorème 01 : Coût de la recherche
Dans une table de Hachage dans laquelle la résolution des collisions se fait par chaînage contenant n clefs est : $\Theta(1+\frac{n}{m})$ dans le cas d'un Hachage uniforme simple

Théorème 02 : Hachage Parfait
Si l'on dispose de n clefs statiques, on peut construire une table de Hachage de taille $O(n)$ telle que la recherche nécessite $O(1)$ opérations dans le pire des cas.

DVP

2. Arbres binaires de Recherche

définition 04 : Arbre binaire de recherche
Soit A un Arbre binaire dont les clefs stockées dans les nœuds appartiennent à un ensemble totalement ordonné.
A est un Arbre Binaire de Recherche ssi
$\forall x$, sommet, $\begin{cases} \forall y \in fg(x), d(x) \leq d(y) \\ \forall y \in fd(x), d(x) \geq d(y) \end{cases}$

exemples : • arbres AVL : un arbre binaire de recherche est un AVL ssi les hauteurs de son fils gauche et de son fils droit ne diffèrent qu'au plus 1.

Proposition 01 : Les opérations d'insertion, de recherche et de suppression dans un AVL se font en temps $O(\ln(n))$

• arbres rouge - noir :
→ toutes les feuilles sont noires ; la racine aussi
→ si un nœud est rouge alors ses enfants sont noirs
→ les chemins simples d'un même sommet à une feuille contiennent tous le même nombre de nœuds noirs.

Proposition 02 : Insérer et supprimer s'exécute en $O(\ln(n))$

Arbres binaires de recherche optimaux :
Soit K, un ensemble de clefs triés et soit D un ensemble de clefs factices représentant les clefs absentes de K.
Un ABRO est un ABR, A, minimisant le coût de recherche espéré : $\mathbb{E}$[coût de recherche dans A] à partir des distributions de K et de D. **Proposition 3 :** Un tel arbre existe et est constructible par

Developpement: Le tri par tas.

Detail de l'algorithme du tri par tas. Reference: Froidevaux - Gaudel - Soria, p 343-353.

(1) Representation par un tas.

① Def On appelle tas un tableau représentant un arbre parfait partiellement ordonné, c'est à dire un arbre dans lequel toutes les feuilles sont situées sur 2 niveaux, avec l'avant dernier niveau complet, et toutes les feuilles du dernier niveau regroupées à gauche (c'est donc un arbre parfait) et donc chaque noeud est plus petit que ses fils. (c'est un arbre partiellement ordonné)

Operation sur un arbre parfait partiellement ordonné.

- adjonction d'un nouvel élément : ajout d'une feuille au dernier niveau de l'arbre, puis reordonner l'arbre en interchangeant noeud et fils si le fils est plus petit que le père.
- suppression du minimum : On supprime la racine, et on la remplace par le dernier noeud. puis on réordonne l'arbre, de même manière que pour l'adjonction d'un element, en changeant avec le fils le plus petit.

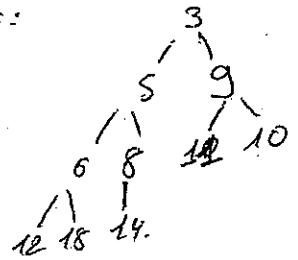
Representation d'un tableau:

On représente un arbre binaire A parfait avec les règles suivantes.

Soit T le tableau représentant A, alors

- T[1] est la racine de A.
- $\forall i$ T[i mod 2] est le père de T[i].
- les deux fils de T[i], s'il existent, sont T[2i] et T[2i+1].

ex:



est representé par le tableau [3; 5; 9; 6; 8; 10; 12; 18; 14]

Un arbre binaire comportant p noeuds est de hauteur $\lfloor \log_2 p \rfloor$. Donc les complexité au pire, en nombre de comparaisons et d'échanges, des operations "adjonction" et "suppression d'un minimum" sont en $\Theta(\log_2 p)$.

en effet, au pire, on doit faire remonter l'element ajouté (resp descendre la nouvelle racine) tout en haut (resp en bas) de l'arbre.

② Le tri par tas proprement dit

② 1ere version Entrée: Une liste a trier, à n éléments

Algorithme ⓘ Construire un tas contenant les n éléments de la liste à trier, par adjonction successives.

D'après le point precedent, on a, au pire $\Theta(\log p)$ comparaison a faire, p le nombre d'éléments déjà dans le tas, et on initialise la liste a trier comme une liste vide

On a alors une complexité, au pire, en $\Theta(n \log n)$ comparaisons

ⓘⓘ On extrait le minimum du tas et on le met à la fin de la liste a trier, cela se fait en temps constant.

ⓘⓘⓘ On supprime le minimum du tas et on reorganise le tas. $\Theta(\log n)$ comparaisons au pire

ⓘⓥ On recommence a l'étape 2 jusqu'a avoir vidé le tas, c'est a dire n fois

② 2eme version

On effectue cette fois un tri en place, c'est a dire directement sur le tableau a trier. dans cette methode, on utilise le debut du tableau pour le tas, et la fin pour les éléments à trier

① Initialisation du tableau par adjonction successive des n éléments

② tant que p (taille du tas) est plus grand que 1, extraire le minimum du tas, et le ranger à la $(p+1)^{\text{eme}}$ place

On a alors un tableau trié dans l'ordre décroissant

Avantage de cette version: Tri en place.

qui coute $\sum_{i=1}^{n} \log_2 i$ comparaisons, au pire.

3eme version : Construction linéaire du tas.

On construit ici le tas en examinant les noeuds, en partant des noeuds les plus bas, et en vérifiant la condition d'ordre partiel avec leurs fils, et en faisant les echanges necessaires.

On ordonne donc successivement des arbres de racine en position $\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor - 1, \dots, 1$.

Cette construction nécessite $\lfloor \frac{n}{2} \rfloor$ appels d'une procédure "ordonner" prenant en entrée p, position du nœud étudié.

Pour chaque appel, on effectue 2 comparaisons, et éventuellement un échange, et éventuellement un appel récursif à "ordonner" pour $j=2p$ et $j=2p+1$ (les fils de $T[p]$).

Majorons le nombre d'appels de la procédure

- si p est compris entre $\lfloor \frac{n}{4} \rfloor + 1$ et $\lfloor \frac{n}{2} \rfloor$, $d \geq n/2$, et donc pas d'appel récursif : 1 appel.
- si p est compris entre $\lfloor \frac{n}{8} \rfloor + 1$ et $\lfloor \frac{n}{4} \rfloor$, au plus 1 appel récursif.

⋮

- si p est compris entre $\lfloor \frac{n}{2^{i+1}} \rfloor$ et $\lfloor \frac{n}{2^i} \rfloor$, on a au plus i appels,

Dans l'intervalle $\left[\lfloor \frac{n}{2^{i+1}} \rfloor + 1, \lfloor \frac{n}{2^i} \rfloor \right]$, on a i appels de la procédure pour chaque élément

On majore grossièrement le nombre d'éléments par $\frac{n}{2^i}$

Donc le nombre d'appels de la procédure est majoré par

$$\sum_{i=1}^{k} \frac{in}{2^i} \leq \frac{n}{2} \sum_{i=1}^{k} \frac{i}{2^{i-1}}$$

et $\sum_{i \geq 1} i x^{i-1}$ est la série dérivée de $\sum x^i$, soit $\frac{1}{1-x}$, donc $\sum_{i \geq 1} i x^{i-1} = \frac{1}{(1-x)^2}$,

que vaut 4 en $x = \frac{1}{2}$. ; qui est son maximum

On majore donc le nombre d'appels par $4\left(\frac{n}{2}\right)$, d'où une construction en $O(n)$