NOM : AUFORT Prénom : William Jury : Kairan Prost

~~Algèbre~~ ← Entourez l'épreuve → ~~Analyse~~ (Informatique)

Sujet choisi : 901 : Structures de données : exemples et applications

Autre sujet : 917 : Logique du premier ordre : syntaxe et sémantique

I] Types et structures de données : introduction

Déf 1 : Un type de données (abstrait) est la description d'un ensemble organisé d'objets et des opérations sur ces objets. Une structure de données est l'implémentation explicite d'un type de données.

Remarque 2 : On s'intéresse ici à la représentation des données, l'implémentation des opérations, mais également leur coût en temps et en espace.

Remarque 3 : Parfois on se placera à des niveaux d'implémentations différents (par exemple Ex 21 et Ex 38).

Objectif 4 : Réaliser efficacement des types de données, pour obtenir avec ces structures des algorithmes performants.

II] Structures séquentielles

Déf 5 : Une liste linéaire est une suite finie d'éléments (e₁, ..., eₙ). On distingue deux cas particuliers : le début et la fin. On souhaite essentiellement insérer et supprimer un élément.

a) Piles et files

Déf 6 : Une pile est une liste linéaire où les insertions et suppressions se font du même côté, appelé le sommet de la pile. Les opérations sur une pile sont :
- EMPILE(x) : insère x au sommet de la pile.
- DEPILE() : enlève l'élément au sommet de la pile.

Implémentation : On utilise un tableau avec un pointeur de sommet de pile et on dit que la pile est vide si le sommet de pile est -1.
- EMPILE(x) revient à déplacer le sommet de pile et mettre l'élément dans le tableau
- DEPILE() revient à renvoyer le sommet de pile et reculer le sommet de pile d'une unité.

Remarque : Comme dans une pile d'assiettes.

Ex 9 :

Ex 10 : On utilise des piles lors de parcours en profondeur d'un graphe.

Déf 11 : Une file est une liste linéaire où l'insertion se fait d'un même côté, et les suppressions se font de l'autre côté. Les opérations sont ENFILER(e) et DEFILER().

Implémentation 12 : On peut utiliser un tableau qui se comporte de manière circulaire : on enfile du début à la fin et quand on arrive à la fin, on enfile au début du tableau. ENFILE revient à avancer la fin de la somme de 1 et DEFILE à avancer le début de la somme de 1.

Ex 13

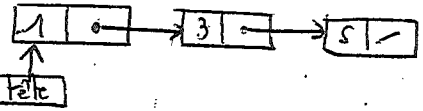
Remarque 14 Cela fonctionne comme une file d'attente.
Remarque 15 : On peut aussi implémenter une file avec 2 piles.
Ex 16 : Utilisé dans des parcours en largeur de graphes.

b) Listes :

Déf 17 Une liste est une liste liéarié où on peut effectuer les insertions / suppressions à l'intérieur.

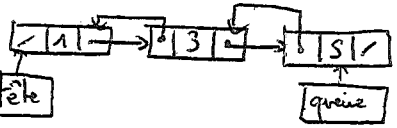
Remarque 18 : On peut alors faire plus d'opérations : parcours, recherche d'un élément, concaténation...

Implémentation 19 : On peut utiliser une liste chaînée : On peut parcourir la liste du début à fin via un système de pointeurs.

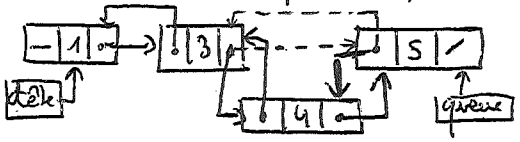
Ex 20 : 

représente [1;3;5]

Implémentation 20 : On peut aussi utiliser une liste doublement chaînée : on peut ainsi parcourir la liste dans les deux sens.

Ex 21 : 

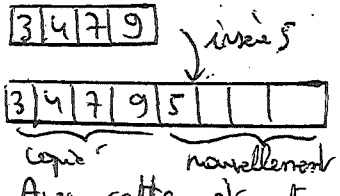
Implémentation 22 Pour insérer un élément à une position donnée, il suffit de construire de nouveaux pointeurs et d'en changer d'autres. Il faut cependant parcourir la liste pour trouver la bonne position.

Ex 23 : Insère 4 après 3 :


Remarque 24 Ainsi, la taille d'une liste est illimitée, ce qui n'était pas le cas pour une pile / file (le tableau la contenant avait une taille fixée.)

c) Tableau dynamique

Idée 25 Quand le tableau est plein et qu'on souhaite insérer un élément au début, on alloue un tableau 2 fois plus grand, on copie les éléments déjà présents, puis on ajoute l'élément.

Ex 26 : 

Remarque 27 Avec cette structure, l'insertion se fait en $O(1)$ amorti.

III Structures arborescentes :

a) Arbre

Déf 28 Un arbre est un type de manière récursive : soit l'arbre est vide, soit il contient un élément appelé racine, ainsi qu'un ensemble d'arbres, ce sont ses fils ou sous-arbres. Lorsqu'un arbre a une racine mais des sous-arbres vides, on dit que c'est une feuille.

Déf 29 Un arbre binaire est un arbre avec au plus 2 fils, qui sont aussi des arbres binaires.

Rem 30 : Des opérations sur les arbres sont l'accès à la racine et aux différents fils, ainsi que la modification de ceux-ci.

Remarque 31 : En spécifiant davantage le comportement voulu pour un arbre, on obtient des types de données différents.

b) Files de priorité / tas :

Déf 32 : Une file de priorité est un type de données opérant sur un ensemble totalement ordonné, qui implémente les fonctions d'insertion, de recherche du minimum, et de suppression du minimum.

Remarque 33 : On peut remplacer min par max.

Déf 33 : Un tas min (resp. un tas max) est une structure de données sous forme d'arbre qui implémente la file de priorité min (resp. max).

Implémentation 34 : Réalisation d'une structure de tas_min avec comme complexité :
- CONSTRUCTION du tas en $O(n)$ (n est le nbre d'éléments du tas).

DEV 1

DEV 1

- Trouver le minimum en $O(1)$
- Supprimer le minimum en $O(\log n)$

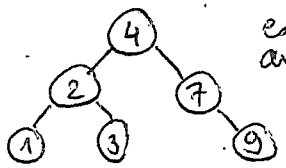
Application 35 : On peut trier un ensemble à n éléments avec un tas min en $O(n \log n)$.

C) Dictionnaires et arbres binaires de recherche

Déf 36 Un dictionnaire est un type de données opérant sur un ensemble totalement ordonné, qui implémente les fonctions de Recherche, d'insertion et de suppression.

Déf 37 Un arbre binaire de recherche est un arbre binaire où, pour chaque nœud de l'arbre, la clé du nœud est supérieure aux clés des nœuds du sous-arbre gauche, et inférieure aux clés des des nœuds du sous-arbre droit.

Ex 38



est un arbre binaire de recherche.

Implémentation 39 :

- Rechercher un élément se fait de manière dichotomique.

```
RECHERCHER (x : noeud, c : clé)
    Si x est vide ou clé(x) = c, retourner x.
    Si c < clé(x)
        RECHERCHER(gauche(x), c)
    Sinon
        RECHERCHER(droit(x), c)
```

- l'insertion fonctionne de la même manière
- la suppression est plus difficile, on veut conserver l'ordre. Si le sommet à supprimer a 2 fils, on le remplace par le sommet le plus à gauche du sous-arbre droit (i.e. le plus petit du sous-arbre droit).

Ces fonctions ont une complexité en $O(h)$, où h est la hauteur de l'arbre.

Remarque 40 : La hauteur d'un arbre binaire de recherche quelconque, peut être n, où n est le nombre d'éléments dans l'arbre.

Implémentation 41 : En ajoutant des hypothèses plus fortes sur la hauteur des sous arbres, on obtient une structure d'arbre Zig-Zag dans lequel les opérations précédentes se font en $O(\log n)$ (on justifiera la complexité et la réalisation de l'insertion dans ce développement).

DEV 2

Remarque 42 On peut imaginer ajouter dans les nœuds des informations plus complexes, comme par exemple une fréquence d'apparition des différents nœuds dans une recherche (par ex. pour traduire un texte). On peut chercher alors à optimiser la durée de la recherche selon cette fréquence d'apparition : on obtient une structure appelé arbre binaire de recherche optimal.

IV Structures relationnelles

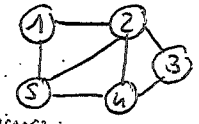
a) Relations entre paires d'éléments : graphes

Déf 43 Un graphe est la donnée d'un ensemble V de sommets et d'un ensemble $E \subseteq V \times V$ d'arêtes. On note $G = (V, E)$.

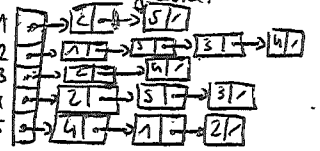
Implémentation 44 : On peut représenter G par une liste d'adjacence : un tableau de taille $|V|$ contenant des pointeurs vers la liste des sommets adjacents au sommet correspondant.

Implémentation 45 : On peut aussi utiliser une matrice d'adjacence : une matrice M telle que $M(u,v) = 1$ si $(u,v) \in E$, 0 sinon.

Ex 46 :



liste d'adjacence :



matrice d'adjacence :

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Remarque 47 : Savoir si $(u,v) \in E$ est en
- $O(|\text{adj}(u)|)$ pour une liste d'adjacence
- $O(1)$ pour une matrice d'adjacence.

D'un autre côté, parcourir toutes les arêtes de G est en $O(|E|)$ pour une liste d'adjacence, mais en $O(|V|^2)$ pour une matrice d'adjacence. Chaque structure est donc adaptée à un usage particulier, et le choix de la structure adaptée à un algorithme est primordial.

Ex 48 : Pour des algorithmes de plus courts chemins de type "all-to-all", comme Floyd-Warshall, on utilise plutôt des matrices, alors qu'en général on préfère les listes d'adjacence, car plus compactes en espace.