IMPLEMENTATIONS ET APPLICATIONS DES ENSEMBLES ET DES DICTIONNAIRES

Intérêts
Comprendre et savoir utiliser les ensembles et les dictionnaires

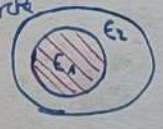
I- Les ensembles

1) Définition
Collection d'éléments non ordonnés et sans doublon

→ défini en énumérant les éléments
ex: $\{0,1\}$, $\{\text{rouge}, \text{noir}\}$, $\mathbb{N} = \{0,1,2,3,\dots\}$

→ défini une propriété
ex: $\{x \in \mathbb{N} \mid x^2 < 10\}$, $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$

2) Opérations et propriétés
· Insertion, Suppression Recherche
· Inclusion: $E \subset E_1$



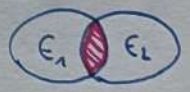
$\forall x \in E \quad x \in E_1$

→ parties de E $\mathcal{P}(E) = \{E' / E' \subseteq E\}$
ex: $\mathcal{P}(\{0,1\}) = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$

· Union: $E = E_1 \cup E_2$

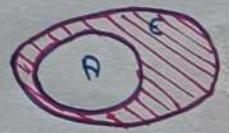
$\forall x \in E \ (x \in E_1 \text{ ou } x \in E_2)$

· Intersection: $E = E_1 \cap E_2$



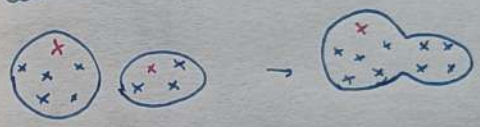
$\forall x \in E \ (x \in E_1 \text{ et } x \in E_2)$

· Complémentaire | $A \subset E$, $B = C_E A$



· Différence $E = E_1 - E_2$

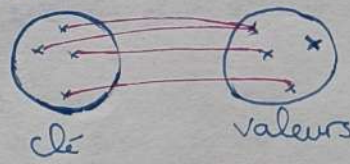
3) Union d'ensemble disjoints : structure UNION-FIND
· S = collection d'ensembles $S = \{S_1, S_2, \dots, S_n\}$
objectif: unir les S_i pour obtenir un ensemble S'
principe: chaque ensemble est représenté par un élément de l'ensemble
ex: délégué d'une classe, racine pour les nœuds d'un arbre,...
· si deux éléments ont le même représentant, ils sont dans le même ensemble
· sinon, on les unit en faisant de l'un le représentant de l'autre



DEV 1 Utiliser la structure union-find pour trouver un arbre couvrant minimal. Algorithme de Kruskal

II- Les dictionnaires

1) Définition
Association d'un ensemble de clés à un ensemble de valeurs.
Chaque clé est unique et associée à une unique valeur.



ex: dictionnaire de définitions

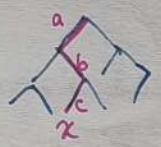
Opérations: insertion
suppression
recherche
modification

2) Arbres

- Représentation de dictionnaires avec des arbres

clé : chemin de la racine à la valeur associé

clé de x : abc



ex : Compression de Huffman
(version statique + alphabet binaire {0,1})

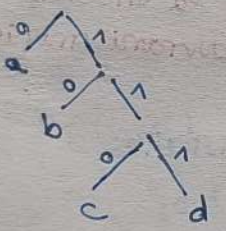
→ objectif : associer à chaque symbole d'un texte une suite de symboles de l'alphabet binaire en minimisant l'espace occupé

→ idée : on calcule la fréquence de chaque symbole, les symboles les plus fréquents sont moins profonds

ex : aabacbad

a : 4
b : 2
c : 1
d : 1

→

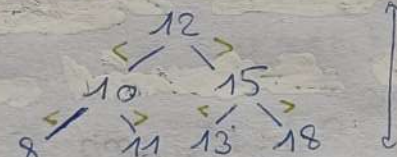


a : 0
b : 10
c : 110
d : 111

⚠ Limiter la hauteur de l'arbre + logique dans la recherche

- Arbre binaires de recherche
  - binaire : tout noeud a 1 ou 2 fils
  - de recherche : pour tout noeud : fils_gauche ≤ noeud.n ≤ fils_droit.n

ex :



h

→ dictionnaire : on stocke un ensemble de paires (clé, valeur)

recherche en $O(h \log h)$ → minimiser h en maintenant des arbres équilibrés ($0 \leq h(A_{gauche}) - h(A_{droite}) \leq 1$)

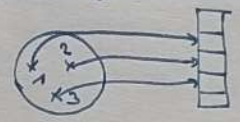
- Généralisation avec des arbres de recherche non binaires équilibrés

DEV 2 B-Arbres : Présentation et opérations

3) Tables de Hachage

- Principe : Les valeurs sont stockées dans un tableau
- Une fonction de hachage calcule l'indice du tableau associé à la clé

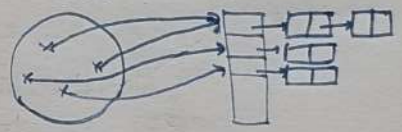
Adressage direct : clé = indice du tableau



⚠ occupation mémoire élevée

Adressage avec liste chaînée

tableau plus petit que l'ensemble des clés
plusieurs clés ↔ un indice du tableau (collision)



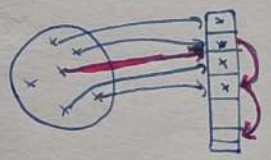
→ meilleure occupation mémoire

→ ⚠ maintenir l'équilibre dans la répartition des clés

recherche d'un élément : calcul + parcours liste chaînée

Adressage ouvert

si on trouve un indice occupé pour une clé, on recalcule un autre indice



⚠ ne pas dépasser du tableau / surcharger la fin

⚠ suppression : ne pas remplacer par NULL

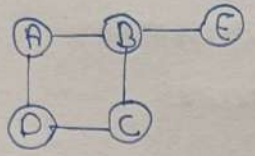
III - Applications

1) Implémentation de structures de données

· **Théorie des graphes :**

implémentation d'un graphe : liste ou matrice d'adjacence
→ si les sommets ne sont pas des entiers : dictionnaires

ex :



Liste d'adjacence

A : {B, D}
B : {A, C, E}
C : {B, D}
D : {A, C}
E : {B}

Matrice d'adjacence

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	0	1
C	0	1	0	1	0
D	1	0	1	0	0
E	0	1	0	0	0

· **Algorithmique du texte :**

· Compression de texte (Huffman, Lempel-Ziv-Welch)
· Recherche de motif dans un texte
→ algorithme de Rabin-Karp : application d'une fonction de hachage au motif.

2) Utilisation en algorithmique

· Enregistrer des résultats associés à des valeur au fur et à mesure du calcul.

→ programmation dynamique :
découpage du problème en sous-problèmes qui se recoupent et se résolvent l'un après l'autre

```
a = tableau
fonction prog_dyn (i):
    si a[i] existe :
        retourner a[i]
    sinon
        a[i] = calcul (i)   ← effectue des appels à prog_dyn
```

3) Problèmes classiques liés aux ensembles

· Recherche d'ensemble intersectant :
$S = \{S_1, \dots, S_n\}$ - $k \in \mathbb{N}$
On cherche H tel que $\begin{cases} \forall i : H \cap S_i \neq \emptyset \\ |H| = k \end{cases}$

→ problème NP-complet

· Somme d'un sous-ensemble
$S = \{x_1, \dots, x_n\}$ $k \in \mathbb{N}$
On cherche $H \subseteq S$ tel que $\sum_{x_i \in H} x_i = k$

→ problème NP-Complet

Conclusion

Structures de données essentielles
→ nombreuses façons de les implémenter
nombreuses applications