Chapitre 1

Structures de données - Bricolage

1.1 Introduction

Le choix d'une structure de données joue souvent un rôle important :

→ pour résoudre intrinséquement des problèmes de structures de données (par exemple, pour une base de données)

→ pour résoudre des problèmes où les structures de données interviennent à des moments-clés d'un algorithme (par exemple, pour des problèmes de graphes)

Cela dépend également du modèle de calcul (pour nous, il sera de type RAM) et du langage de programmation (on utilisera par la suite du pseudo-code de type C). L'idéal est de récupérer une structure de donnée bien connue, déjà analysée et implémentée.

1.2 Faire soi-même sa structure de données

Définition 1.1 Une partition $\mathcal{P}$ d'un ensemble $\mathcal{E}$ est une famille $(X_i)_{i\in[\![1,k]\!]}$ de parties disjointes de $\mathcal{E}$ telle que : $\bigcup\limits_{i\in[\![1,k]\!]} X_i = \mathcal{E}$.

Définition 1.2 Soient $\mathcal{P}$ et $\mathcal{Q}$ deux partitions de $\mathcal{E}$.

On dit que $\mathcal{P}$ est plus fine que $\mathcal{Q}$ et on note $\mathcal{P} \leqslant \mathcal{Q}$ si $\forall X \in \mathcal{P}, \exists\, Y \in \mathcal{Q}$ tel que $X \subset Y$ (on dit également que $\mathcal{Q}$ est plus grossière que $\mathcal{P}$) (voir Fig. 1.1).

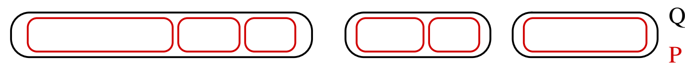


Fig. 1.1 – *$\mathcal{P}$ plus fine que $\mathcal{Q}$*

Définition 1.3 *Affinage*

Soient $\mathcal{P}$ une partition de $\mathcal{E}$ et soit $S \subset \mathcal{E}$.

$\mathcal{P}' = \texttt{Affiner}(\mathcal{P},\mathcal{S})$ est la partition obtenue en remplaçant dans $\mathcal{P} = (X_i)_{i\in[\![1,k]\!]}$ les parties X_i intersectant strictement S par les deux sous-parties $X_i' = X_i \cap S$ et $X_i'' = X_i \backslash S$ (voir Fig. 1.2).

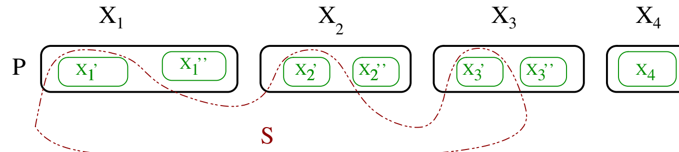


FIG. 1.2 – *Affinage*

Remarque 1.1 `Affiner`$(\mathcal{P}, S) \leqslant \mathcal{P}$.

1.2.1 Objectif

On veut trouver une structure de données pour $\mathcal{P}$ telle que pour tout S, on peut calculer `Affiner`$(\mathcal{P}, S)$ en $O(|S|)$, avec l'hypothèse que l'on peut accéder en $O(|S|)$ aux position de tous les éléments de S dans la structure de données pour $\mathcal{P}$ (c'est-à-dire qu'on peut y accéder en $O(1)$ en temps amorti).

Pour satisfaire cette hypothèse, on peut par exemple tenir à jour une liste de pointeurs sur les positions des éléments de S dans $\mathcal{P}$ (voir FIG. 1.3).

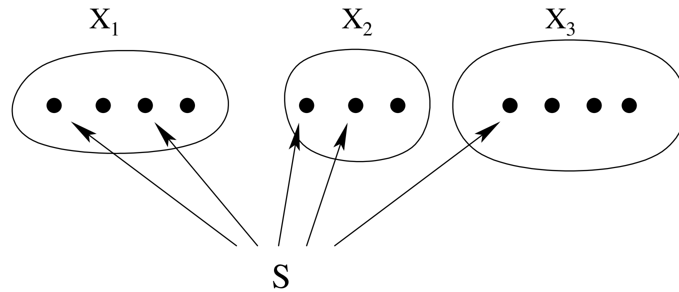


FIG. 1.3 – *Hypothèse*

Exemple

$\mathcal{E} = \{x_1; ...; x_8\}$

$\mathcal{P} = \{\{x_1; x_2; x_3\}; \{x_4; x_5\}; \{x_6; x_7; x_8\}\}$

$S = \{x_1; x_3; x_5; x_7\}$

On numérote les partitions tout en tenant à jour un entier n contenant le plus grand des numéros des partitions. Ainsi, dans notre exemple :

$\mathcal{P}$	$\mathcal{P}'$
$(x_1, 1)$	$(x_1, 4)$
$(x_2, 1)$	$(x_2, 1)$
$(x_3, 1)$	$(x_3, 4)$
$(x_4, 2)$	$(x_4, 2)$
$(x_5, 2)$	$(x_5, 5)$
$(x_6, 3)$	$(x_6, 3)$
$(x_7, 3)$	$(x_7, 6)$
$(x_8, 3)$	$(x_8, 3)$
$n = 3$	$n = 6$

1.2.2 Objectifs supplémentaires

On veut de plus :

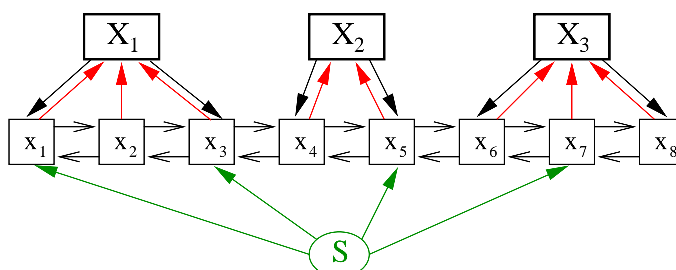
→ tester en $O(1)$ si deux éléments sont dans la même partie (ce qui est vérifié dans l'exemple précédent)

→ lister les éléments :

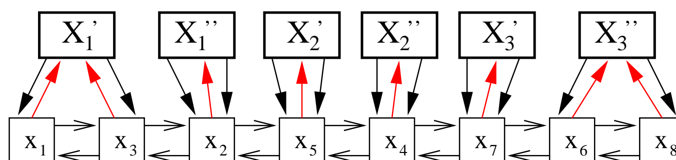
- d'une même partie X_i
- dans la même partie X_i q'un élément x

en $O(|X_i|)$ (ce qui n'est pas vérifié dans l'exemple précédent)

Pour cela, on peut par exemple utiliser une liste doublement chaînée pour représenter les éléments de $\mathcal{E}$. De plus, chaque élément pointe vers le représentant de sa partition, qui pointe lui-même vers les premier et dernier éléments de la partition. Affiner revient alors à mettre "à gauche" les éléments de $X_i \cap S$ (voir Fig. 1.4).



(a) *Avant*



(b) *Après*

Fig. 1.4 – *Exemple de structure de données*

1.2.3 Application

On va appliquer l'affinage pour le calcul de sommets jumeaux dans un graphe.

Définition 1.4 Soit $G = (V, E)$ un graphe. On note $\mathcal{N}(u) = \{v \in V / \{u; v\} \in E\}$ l'ensemble des voisins de u pour tout $u \in V$ (voir FIG. 1.5).

On dit que u et v sont jumeaux ssi $\mathcal{N}(u) = \mathcal{N}(v)$.

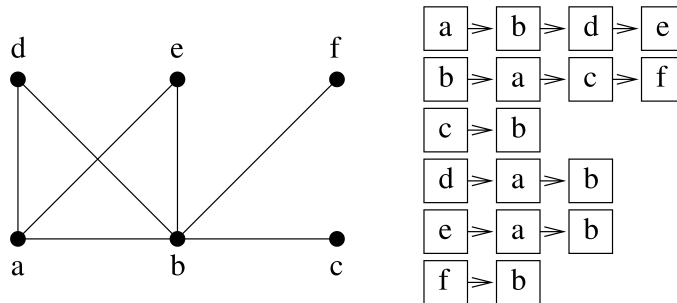


FIG. 1.5 – *Voisins dans un graphe*

Remarque 1.2 En ajoutant la réflexivité, ceci est une relation d'équivalence.

Objectif : Etant donné un graphe $G = (V, E)$ donné par ses listes d'adjacence, calculer les classes d'équivalence.

Remarque 1.3 Pour un graphe non orienté, on peut trier les listes d'adjacence en $O(|V| + |E|)$.

Première méthode Pour chaque nouveau sommet, on teste s'il est jumeau avec un représentant de chaque classe déjà calculée (après avoir trié les listes d'adjacence en choisissant un ordre arbitraire sur V).

Le coût de cet algorithme est :

$$\sum_{\{u;v\} \in E} \text{Min}\{\deg(u); \deg(v)\}$$

Donc, dans le pire des cas, le coût est en :

$$\frac{n(n-1)}{2} \theta(n) = \theta(n^3)$$

Deuxième méthode On réalise un affinage en partant de $\mathcal{P} = \{V\}$ et avec les ensembles $\mathcal{N}(u)$ pour $u \in V$.

→ Complexité : en $O(|V| + |E|)$.

→ Correction : u n'est pas jumeau de v ssi $\exists\, w \in \mathcal{N}(u) \backslash \mathcal{N}(v)$ ou $\mathcal{N}(u) \backslash \mathcal{N}(v)$; ce w va séparer u et v au cours des affinages successifs.

1.2.4 Autres applications

→ `Quicksort` peut être vu comme de l'affinage

→ Tri lexicographique de mots

→ Algorithme de Hopcroft de minimisation d'automates