TD Info 6 : Union-Find

Michael Monerau

13 janvier 2011

1 Problème formel

On se fixe comme objectif pour ce TD de mettre au point une technique permettant de grouper n objets en sous-ensembles distincts, tout en pouvant retrouver efficacement à quel groupe un élément particulier appartient.

Il convient de définir correctement les opérations de base que l'on veut implémenter, trouver les algorithmes correspondant se fera dans une seconde étape. C'est une démarche habituelle : on définit ici l'*interface* de notre "structure de données ensemble". L'implémentation de celle-ci ne concerne pas l'appelant, on lui assure que la description des opérations est respectée. Charge à nous ensuite d'implémenter les opérations le plus efficacement possible.

On suppose que l'on veut stocker des éléments distincts deux à deux. Une *structure de données ensemble* maintient une collection $\mathcal{S} = (S_1, \ldots, S_k)$ d'ensembles, ce sont les ensembles manipulés par l'implémentation, où $S_i \subseteq U$ et U est l'univers comportant les objets. Chacun des S_i est identifié par un *représentant* qui est simplement un élément de S_i, a priori sans contrainte particulière en général mais qui ne change pas si la structure n'est pas modifiée par l'extérieur.

On demande à ce que la structure de données implémente trois opérations de base :

- Make-Set(x) : où $x \in U$. Crée un nouvel ensemble S_i qui ne contient que x. x est donc son représentant.
- Union(x, y) : fusionne les deux ensembles qui contiennent x et y, disons S^x et S^y, en un nouvel ensemble qui est l'union $S^x \cup S^y$. Le représentant est une élément de $S^x \cup S^y$. Noter que S_x et S_y ne sont plus dans $\mathcal{S}$ après l'opération.
- Find-Set(x) : renvoie le représentant de l'ensemble contenant x.

Pour les calculs de complexité, on dénotera par n le nombre d'opérations Make-Set, et par m le nombre de n'importe laquelle des trois opérations ci-dessus.

On suppose d'autre part que les n Make-Set sont les premières actions effectuées (on construit $\mathcal{S}$ pour commencer, c'est raisonnable).

Question 1.1. Formalisme
Voir qu'on a donc au plus $n-1$ opérations Union, et que $n \leq m$.

Comment savoir si deux éléments sont dans le même ensemble ?

2 Implémentation de l'interface

On pourra supposer que les éléments sont des entiers, quitte à indexer les vrais éléments par des entiers. On supposera également pour simplifier qu'ils sont en nombre fini $N = 15$.

2.1 Représentation par listes

2.1.1 Principe

Une implémentation directe de l'interface est de stocker pour chaque S_i l'ensemble de ses éléments sous la forme d'une liste. Son représentant peut alors être pris comme la tête de la liste.

Question 2.1. Implémentation
Implémenter les trois opérations de base en utilisant cette vision.

Indication : On pourra utiliser des types record comme éléments d'un `vect` pour tenir à la main une liste qui stocke à la fois les valeurs et leur représentant en cours.

2.1.2 Complexité

Question 2.2. Complexité
Quelle est la complexité des opérations implémentées ci-dessus ?

Pour accélérer l'opération d'Union, on choisit à chaque fois de concaténer la plus grande à la plus

courte. Pourquoi est-ce que c'est intéressant (on suppose que la longueur d'une liste se calcule en $O(1)$, ce qui peut être fait en conservant un champ `taille` dans la liste en interne) ?

Théorème 1. *Avec cette heuristique d'union pondérée, l'exécution des m requêtes se fait en temps $O(m + n \ln n)$.*

Démonstration. La preuve est un comptage assez précis, cf. au tableau. □

2.2 Représentation par forêt

2.2.1 Principe

Cette fois-ci, les ensembles sont représentés par des arbres. Le représentant est la racine. Plutôt que de représenter les arbres comme à notre habitude, on stocke ici simplement pour chaque noeud/feuille son père.

Question 2.3. Implémentation
Implémenter les trois opérations de base en utilisant cette vision.

2.2.2 Complexité

Question 2.4. Complexité
Que dire de la complexité des opérations dans ce nouveau cadre ?

2.2.3 Amélioration

Il y a deux améliorations possibles sur ces algorithmes pour parvenir à l'implémentation la plus rapide connue.

Union by rank

L'idée est de fusionner le plus petit arbre sur le plus gros, comme on avait fait ci-dessus avec les listes. Plutôt que de retenir pour chaque noeud/feuille la hauteur de son sous-arbre, on assigne plutôt un *rang* à chaque noeud, qui est une borne supérieure sur la hauteur du noeud. Pour cela, on assigne un rang de 0 lors de Make-Set, puis on incrémente de 1 lorsqu'on fait un Union qui augmente la hauteur de 1.

Question 2.5. Implémentation
Implémenter cette amélioration.

Path compression

L'idée est de faire pointer tous les nœuds traversés lors d'une recherche *directement* vers la racine plutôt que de passer par les nœuds intermédiaires. Cependant, cette opération ne modifie pas les rangs des nœuds (pourquoi ?).

Question 2.6. Implémentation
Implémenter cette amélioration.

On peut alors montrer le :

Théorème 2. *Avec ces deux améliorations, la complexité des opérations est quasi-linéaire en m : $O(m\ \alpha(n))$ où α est une fonction qui croît* très *lentement (≤ 4 pour des n pratiques, ie. $n \leq 10^{80}$).*

3 Applications

Beaucoup d'algorithmes utilisent Union-Find, souvent même sans le dire comme brique de base classique. La linéarité en pratique est évidemment très attirante. Voyons trois tels algorithmes.

3.1 Connexité dans un graphe

Question 3.1. Connexité
Supposons qu'on donne un graphe comme la liste de ses arêtes. Donner un algorithme utilisant Union-Find qui permet de dire si deux sommets sont ou non dans la même composante connexe.

Donner un avantage de cette approche sur l'algorithme déjà vu au TD 2 de parcours DFS ou BFS du graphe.

3.2 Plus petit ancêtre commun dans un arbre

Étant donné un ensemble de paires de nœuds P, on veut trouver leur ancêtre commun qui leur est le plus proche (le plus bas dans l'arbre). [Voir explications au tableau]

3.3 Équivalence de deux automates finis

L'algorithme de Hopcroft Karp permet de déterminer si deux automates finis sur le même alphabet reconnaissent ou non le même langage. [Voir explications au tableau]