L3-INFO Algorithmique

Université Joseph Fourier GRENOBLE UFR IM²AG

ALGO5 : Travaux dirigés, séance 6
Union-Find

On considère un ensemble E de n éléments identifiés par les entiers de 1 à n et une partition P définie sur E par une relation d'équivalence R.

Rappel : $(i, j) \in R \Leftrightarrow i$ et j sont dans la même classe.

On veut réaliser les opérations suivantes :
- `MemeClasse (i,j)` est vrai si et seulement si i et j sont dans la même classe de P, i.e. $(i, j) \in R$.
- `Find(i)` est la classe de l'élément i. Cette opération n'est définie que si la représentation des noms de classe est explicite. Dans ce cas,
  `MemeClasse (i,j) = ((i=j) ou Find(i)=Find(j))`.
- `Union(i, j)` fusionne les classes de i et de j ce qui modifie P.

Ces opérations servent à construire une partition à partir d'un ensemble couples SC représentant des éléments en relation. Initialement on a une classe par élément. Pour chaque couple $(i, j) \in SC$, si i et j ne sont pas dans la même classe alors fusionner les classes de i et de j. On va ainsi réaliser une suite de k ($k \leqslant n$) opérations `Find` et `Union`.

Travail préliminaire

E = { 1, 2, 3, 4, 5, 6, 7, 8 }
SC = { (3,6), (2,5), (1,3), (4,7), (8,1), (5,8) }
Initialement, P = { {1} , {2}, {3}, {4}, {5}, {6}, {7}, {8} }.
Après la première étape qui correspond au traitement du couple (3,6) :
P = { {1} , {2}, {3, 6}, {4}, {5}, {7}, {8} }.

Question 0.1 :

Poursuivre le déroulement de l'agorithme.

Nous allons étudier des structures de données différentes pour représenter la relation d'équivalence et évaluer dans chaque cas le coût des opérations `Find` et `Union`. Noter que ce qui nous intéresse ici est le coût de l'application de k ($k \leqslant n$) opérations `Find` et `Union`.

Union-Find

1 Représentation des classes par des vecteurs de booléens

Chaque classe de la relation est représenté par un vecteur de booléens. La partition est représenté par un tableau des classes.

```
NbC : entier sur [1..n]      { nombre de classe }
P : tableau sur [1..n] de (tableau sur [1..n] de booleen)
      { P[i,j] est vrai
        ssi
        j est dans la classe representee par la ligne i }
```

Les indices de ligne définissent les noms des classe, les indices de colonnes représentent les noms des éléments.

Question 1.1 : Etude de propriété

Dans la suite on suppose la propriété suivante : pour un élément j donné, il existe un et un seul k tel que $P[k, j]$.

Que signifie cette propriété ? Que pouvez vous dire concernant les valeurs remplissant le tableau ?

Question 1.2 : Find(j)

Ecrire le code de la fonction Find. Compter le nombre d'accès à P.

Question 1.3 : Union(j1, j2)

Ecrire le code de la fonction Union. Compter le nombre d'accès à P.

2 Tabulation de la fonction Find (quick Find)

Version 1

On associe à chaque élément son numéro de classe. Soit le tableau CL tel que `CL[i]` est le numéro de classe de l'élément i :
`CL : tableau sur [1..n] d'entiers sur [1..n].`

Question 2.1 : Find(i)

Ecrire le code de la fonction Find. Compter le nombre d'accès à CL.

Question 2.2 : Union(i, j)

Ecrire le code de la fonction Union. Compter le nombre d'accès à CL.

Question 2.3 : k opérations Union

Quelle est l'ordre de grandeur de la complexité de k ($k \leqslant n$) opérations d'union ?

Pour réduire cette complexité une idée est de faire en sorte que lors d'une opération d'union on intègre toujours la classe la plus petite dans la plus grande. Il faut ainsi modifier les noms de classe des éléments de la plus petite des classes données ; pour cela on associe à chaque classe i son nombre d'éléments nb_i ($1 \leqslant nb_i \leqslant n$).

Question 2.4 : Union(i, j)

Ecrire le code de la fonction Union ainsi modifiée. Quelle est la complexité d'une suite de k opérations d'union ?

3 Représentation des classes par des listes chaînées (quick Union)

Tout élément n'appartenant qu'à une classe, les listes peuvent être représentées dans un seul tableau `Suiv`, `Suiv[i]` étant l'élément successeur de i dans la liste correspondant à la classe de i. De plus, les classes n'étant pas vides, ces listes peuvent être rendues circulaires.

On considère la déclaration :

```
Suiv : tableau sur [1..n] d'entiers sur [1..n]
```

Question 3.1 : MemeClasse(i, j)

Ecrire le code de la fonction `MemeClasse (i, j)`. Compter le nombre d'accès au tableau `Suiv`.

Question 3.2 : Union(i, j)

Ecrire le code de la fonction Union ainsi modifiée. Quelle est sa complexité ?

4 Représentation des classes par des arbres

Une classe est représentée par un arbre n-aire, le nom de la classe étant celui de l'élément racine de l'arbre. L'union de deux classes consiste à placer l'un des arbres en sous-arbre de la racine de l'autre. Trouver la classe d'un élément consiste en un parcours du chemin allant de cet élément à la racine de son arbre et a ainsi une complexité fonction de la hauteur de l'arbre.

Comme tout élément de l'ensemble appartient à une seule classe donc à un seul arbre, on peut représenter les arbres dans un tableau par un codage de la fonction père.

version 1

On considère la déclaration :

```
P : tableau sur [1..n] d'entiers sur [0..n]
```

où `pere[i]` est le père de i dans l'arbre représentant la classe de i ou 0 si i est une racine.

Question 4.1 : Find(i)

Ecrire le code de la fonction Find. Compter le nombre d'accès à P.

Question 4.2 : Union(i, j)

Ecrire le code de la fonction Union. Compter le nombre d'accès à P.

Question 4.3 : k opérations Find(i)

Quelle est la complexité de k opérations `Find(i)` ?

version 2 : weighted quick Union, compression path

Pour diminuer la complexité lors de l'enchainement de k opérations, il faut minimiser la hauteur des arbres. Lors d'une union l'idée est de placer en sous-arbre de l'autre celui des deux qui a la hauteur la plus faible. Pour cela il faut mémoriser la hauteur de tout arbre ; on stocke la valeur $-h_i$, h_i étant la hauteur de l'arbre de racine i, dans `P[i]` au lieu de stocker 0.

Question 4.4 : Find(i)

Quelle est alors la complexité de l'opération Find(i) ?

Une technique supplémentaire pour raccourcir la hauteur des arbres consiste, lors de l'opération Find, à raccrocher à la racine tous les sous-arbres du chemin conduisant de la racine à l'élément considéré. Les arbres traités deviennent ainsi très plats. La complexité d'une suite d'opérations find et union devient alors quasi linéaire.

Question 4.5 : Find(i)

Ecrire le code de la fonction Find en utilisant cette technique.

Question 4.6 : Union(i, j)

Ecrire le code de la fonction Union en utilisant cette technique.