Ensembles disjoints

CHAPITRE 8 (WEISS)

Ensembles disjoints

- Définition des ensembles disjoints
  - Opérations de base : créer un ensemble, trouver l'ensemble représentant un élément, unir deux ensembles
  - Représentation des ensembles disjoints : listes chaînées, arbres

Algorithmes de manipulation des ensembles disjoints

- Algorithme de recherche de l'ensemble représentant un élément
- Algorithme d'union de deux ensembles
- Optimisation des opérations avec les heuristiques de rang et de compression de chemin

Ensembles disjoints

Applications des ensembles disjoints

- Détection de cycles dans un graphe
- Regroupement et clustering de données
- Algorithme de Kruskal pour l'arbre couvrant de poids minimal

Complexité et analyse des performances

- Complexité temporelle des opérations sur les ensembles disjoints

Définition

- Les ensembles disjoints représentent une structure de données utilisée pour regrouper des éléments en ensembles distincts, où chaque élément appartient à un seul ensemble
  - Les ensembles disjoints sont caractérisés par le fait que deux ensembles ne peuvent pas avoir d'éléments en commun
  - Ils sont également connus sous le nom d'ensembles disjointifs ou de structures disjointes

Relations d’équivalence

- Ensembles disjoints – structure efficace pour résoudre des problèmes d'équivalence
- Une relation R est définie sur un ensemble S si, pour chaque paire d'éléments (a, b) (a, b ∈ S) a R b est soit vrai, soit faux
- Si a R b est vrai, alors nous disons que a est en relation avec b.
- Relation d’équivalence R satisfait à trois propriétés :
  - (Réflexivité) a R a, pour tout a ∈ S
  - (Symétrie) a R b si et seulement si b R a
  - (Transitivité) a R b et b R c implique a R c

Relations d’équivalence

- Relation d’équivalence R satisfait à trois propriétés :
  - (Réflexivité) a R a, pour tout a ∈ S
  - (Symétrie) a R b si et seulement si b R a
  - (Transitivité) a R b et b R c implique a R c
- Exemples
  - ≤ n'est pas une relation d'équivalence (n'est pas symétrique, car a ≤ b n'implique pas b ≤ a)
  - connectivité électrique, où toutes les connexions se font par des fils métalliques, est une relation d'équivalence

Problème d'équivalence dynamique

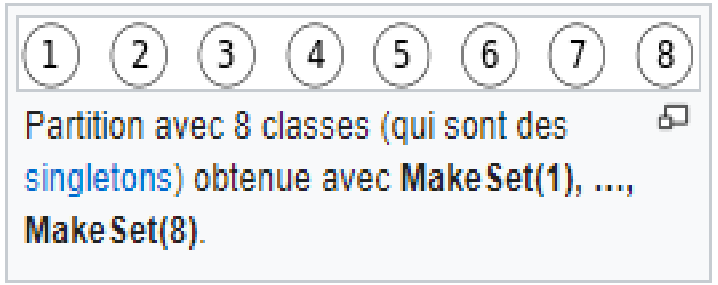
- Étant donnée une relation d'équivalence ~, décider, pour tout a et b, si a ~ b
  - Si la relation est stockée sous forme d'un tableau bidimensionnel de variables booléennes cela peut être fait en temps constant
- Le problème de relation est généralement défini de manière implicite plutôt qu'explicitement
- Exemple : relation d'équivalence soit définie sur l'ensemble à cinq éléments {a1, a2, a3, a4, a5} =>
  - Il y a 25 paires d'éléments, chacune étant soit en relation, soit non en relation
  - Cependant, les informations a1 ~ a2, a3 ~ a4, a5 ~ a1, a4 ~ a2 impliquent que toutes les paires sont en relation
- Nous aimerions pouvoir en déduire cela rapidement.

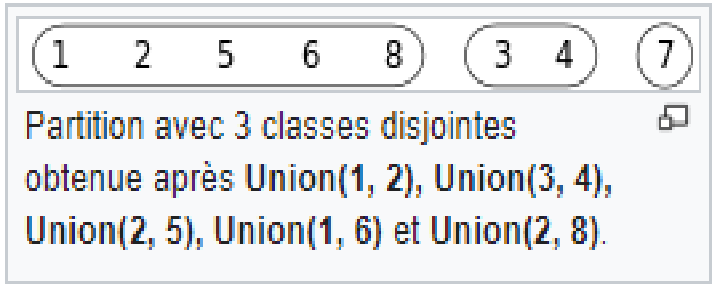
Ensembles disjoints

- Une **classe d'équivalence** d'un élément $a \in S$ est le sous-ensemble de S qui contient tous les éléments en relation avec a
- Classes d'équivalence forment une partition de S : chaque élément de S apparaît exactement dans une classe d'équivalence
- Pour décider si $a \sim b$, nous devons simplement vérifier si a et b se trouvent dans la même classe d'équivalence
- Cela nous donne notre stratégie pour résoudre le problème d'équivalence

Ensembles disjoints

- Au départ, l'entrée est une collection de N ensembles, chacun avec un seul élément. Cette représentation initiale indique que toutes les relations (à l'exception des relations réflexives) sont fausses
- Chaque ensemble a un élément différent, de sorte que Si ∩ Sj = ∅ ; cela rend les **ensembles disjoints**
- Les opérations de base sur les ensembles disjoints comprennent
  - création d'un nouvel ensemble
  - recherche de l'ensemble contenant un élément donné
  - union de deux ensembles en un seul





https://fr.wikipedia.org/wiki/Union-find

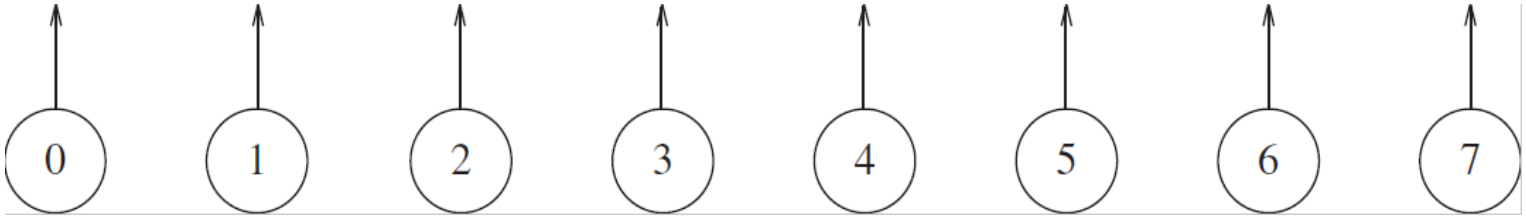
Structure de donnée UNION_FIND

- Une structure de données d'ensembles disjoints est une structure de donnée qui maintient à jour une collection S = { S1, S2, …, Sk } d'ensembles dynamiques disjoints
- Chaque ensemble est identifié par un représentant qui est un certain membre de l'ensemble.

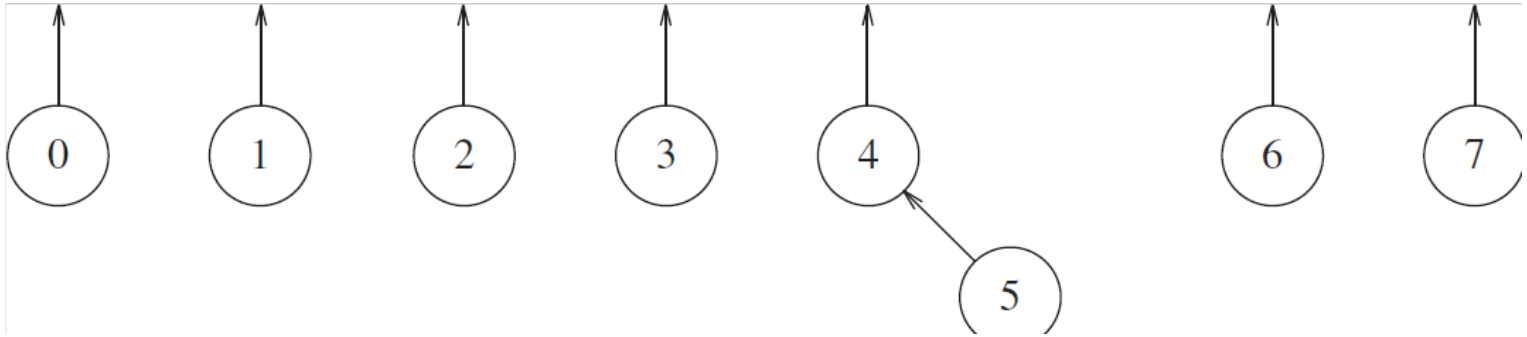
Opérations de base

- **Création d'un nouvel ensemble (CRÉER-ENSEMBLE(x))**
- **Find (TROUVER-ENSEMBLE(x))** (recherche de l'ensemble contenant un élément donné)
  - Renvoie le nom de l'ensemble contenant un élément donné
- **Union(UNION(x,y)) =>** Ajouter des relations
  - Si nous voulons ajouter la relation a ~ b, alors nous vérifions d'abord si a et b sont déjà en relation
  - Cela se fait en effectuant des « finds » à la fois sur a et b et en vérifiant s'ils se trouvent dans la même classe d'équivalence. S'ils ne le sont pas, alors nous appliquons l'union
  - Union fusionne les deux classes d'équivalence contenant a et b en une nouvelle classe d'équivalence => créer Sk = Si U Sj, en détruisant les ensembles originaux et en préservant la disjonction de tous les ensembles

Ensembles disjoints

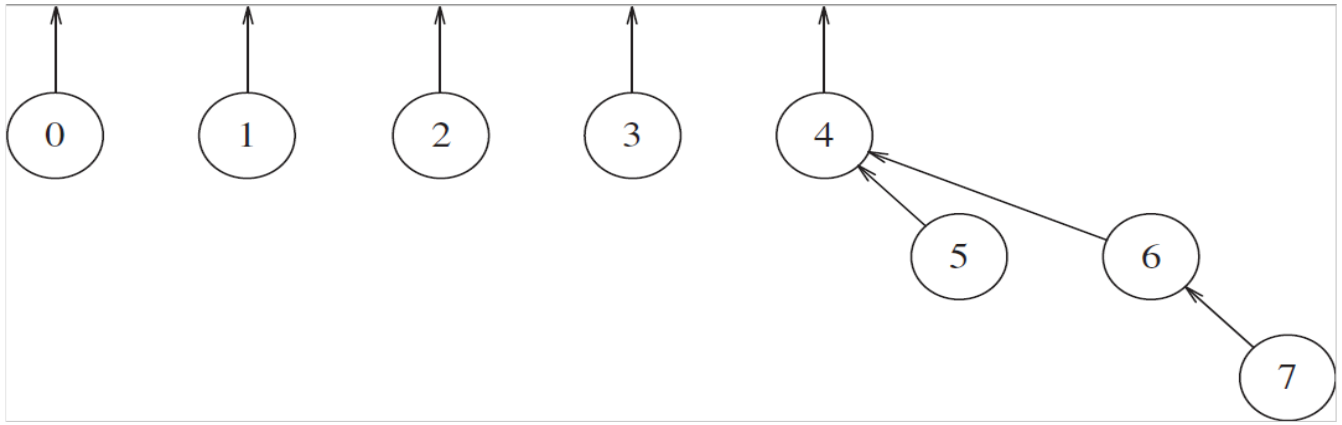


Exemple : 8 éléments initialement placés dans les différents ensembles



Après union(4,5)

Ensembles disjoints



Après union(6,7) et (4,6)

–1	–1	–1	–1	–1	4	4	6
0	1	2	3	4	5	6	7

Pour chaque ensemble i on sauvegarde dans la case i d'un tableau l'index de son parent

Opérations de base, TROUVER-ENSEMBLE(x)

- Aucune opération de comparaison sur des valeurs
- Nous avons besoin de savoir des relations d'appartenance d'une valeur à un ensemble donné
- Le nom retourné par TROUVER-ENSEMBLE(x) n'a pas d'importance dans le sens direct
- Important est de pouvoir tester les relations entre les éléments
  - TROUVER-ENSEMBLE(a) == TROUVER-ENSEMBLE(b) est vrai si et seulement si a et b se trouvent dans le même ensemble

Représentation des ensembles disjoints

- Listes chaînées
- Arbres

Représentation des ensembles disjoints

- **Listes chaînées**
- Chaque ensemble est représenté par une liste chaînée.
- Le premier objet de chaque liste chaînée sert de représentant
- Chaque objet de la liste chaînée contient un élément de l'ensemble, un pointeur sur l'objet contenant l’élément suivant, et un pointeur sur le représentant de l'ensemble

Analyse du temps

- Représentation en liste chaînée
  - CREER ENSEMBLE, et TROUVER ENSEMBLE sont simples a implémenter, et consomment O(1) au pire des cas
  - L'implémentation la plus simple de l'opération UNION(x; y), exécutée par concaténation de la liste de x à la fin de la liste de y, consomme beaucoup plus de temps, même Θ(n) au pire des cas
  - Heuristique d'union pondérée
    - La représentation par listes chaînées demande Θ(m) en moyen par appel (on pourrait concaténer la liste la plus longue (taille m) à la plus courte ; Dans ce cas il faut mettre à jour le pointeur sur le représentant pour chaque membre de la liste la plus longue

Analyse du temps, union

- Représentation en liste chaînée
  - Heuristique d'union pondérée
    - Supposons que chaque représentant contient également la longueur de la liste
    - On concatène toujours la plus petite liste à la plus longue
  - Union peut prendre encore Θ(m) temps si les deux ensembles ont Θ(m) éléments

Représentation des ensembles disjoints

- Arbres (Fôret d'ensembles disjoints)
- Chaque ensemble – un arbre enraciné
- La racine de chaque arbre contient le représentant
- Chaque nœud contient un élément de l'ensemble et un pointeur sur son père
- Pour chaque racine le pointeur sur le père pointe sur la racine soi-même

Implémentation des opérations (Arbres)

- Créer un ensemble – créer un arbre à un seul nœud
- Trouver ensemble – suit les pointeurs pères jusqu'à rencontrer la racine de l'arbre
  - Les nœuds visités sur ce chemin constituent la route directe
- Union fait pointer la racine d'un arbre sur la racine de l'autre

Opération Union

- Utilisant 2 heuristiques on peut atteindre un temps d'exécution qui est presque linéaire par rapport au nombre total d'opérations m
- Union par rang
- minimiser la hauteur de l'arbre pour obtenir une petite longueur de la route directe pour un nœud quelconque de l'arbre
- Pour chaque nœud, on maintient à jour un rang(x) qui est une approximation du logarithme de la taille du sous-arbre enraciné à x, et qui est aussi une borne supérieure pour la hauteur du sous-arbre enraciné à x
- Il n'est pas nécessaire de mettre à jour la hauteur de sous-arbre enraciné à x pour chaque nœud (stratégie paresseuse)

Opération Union par rang

- Union par rang
  - Toujours attacher l'arbre le plus petit à la racine de l'arbre le plus grand
  - Pour évaluer quel arbre est le plus grand, on utilise une heuristique appelée le rang
  - Arbres contenant un élément sont de rang zéro
  - Lorsque deux arbres de même rang sont réunis, le résultat a un rang plus grand d'une unité
  - Avec cette seule amélioration, la complexité amortie : des opérations CRÉER-ENSEMBLE, UNION et TROUVER-ENSEMBLE devient O(log n) et O(1) au pire cas et au meilleur cas

Amélioration de « find » avec la compression de chemin

- Profiter de chaque **TROUVER-ENSEMBLE** pour aplatir la structure d'arbre
  - Chaque nœud rencontré sur le chemin menant à la racine peut être directement relié à celle-ci car tous ces nœuds ont le même ancêtre
- Pour réaliser cela:
  - On fait un parcours vers la racine, afin de la déterminer
  - Puis un autre parcours pour faire de cette racine le parent de tous les nœuds rencontrés en chemin
  - L'arbre résultant ainsi améliore les performances des futures recherches d'ancêtre, mais profite aussi aux autres nœuds pointant vers ceux-ci, que ce soit directement ou indirectement

Implémentation d'opérations

CRÉER-ENSEMBLE(x)

1. P[x] ← x
2. Rang[x] ← 0

UNION(x, y)

1. LIER(TROUVER-ENSEMBLE(x), TROUVER-ENSEMBLE(y))

Implémentation d'opérations

LIER(x,y)

1. Si rang[x] > rang[y]
2. alors p[y] ← x
3. sinon p[x] ← y

4. Si rang[x] == rang[y]

5. Alors rang[y] ← rang[y] + 1

Implémentation d'opérations

TROUVER-ENSEMBLE(x) ;

1. Si $x \neq p[x]$
2. alors $p[x] \leftarrow$ TROUVER-ENSEMBLE(p[x])
3. retourner p[x]

Deux phases :

1. Remonter le long de la route directe jusqu'à la racine
2. Redescendre la route directe pour mettre à jour chaque nœud de manière qu'il pointe directement sur la racine (compression de chemin)

Conclusion

- Les ensembles disjoints trouvent de nombreuses applications dans divers domaines, tels que l'analyse de graphes, la gestion des relations d'équivalence, la détection de cycles, le regroupement de données et la résolution de problèmes d'optimisation. Ils offrent une efficacité accrue dans la manipulation de collections d'éléments et permettent d'effectuer des opérations telles que l'union rapide et la recherche d'appartenance en temps logarithmique ou quasi-constant.
- En résumé, les ensembles disjoints représentent une structure de données fondamentale permettant de regrouper des éléments en ensembles distincts sans chevauchement, offrant des opérations efficaces pour la manipulation et l'analyse de collections d'éléments.