Estructuras para conjuntos disjuntos

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información

Plan

Preliminares

Detectando componentes conexas con DFS

Ejercicio 22.3-12 de [1]

Show that we can use a depth-first search of an undirected graph G to identify the connected components of G, and that the depth-first forest contains as many trees as G has connected components. More precisely, show how to modify depth-first search so that it assigns to each vertex v an integer label $v.cc$ between 1 and k, where k is the number of connected components of G, such that $u.cc = v.cc$ if and only if u and v are in the same connected component.

Detectando componentes conexas con DFS

Procedimiento componenteConexaDFS(G =(V, E)) G es no dirigido

```
inicio
    para cada (v ∈ V) hacer v.color ← BLANCO; v.pred ← NIL ;
    tiempo ← 0 ; /* Variable global                                    */
    contCC ← 0 ; /* Variable global contador de Comp. Conexas          */
    para cada v ∈ V hacer
        si (v.color = BLANCO) entonces
            contCC ← contCC + 1 ;
            dfsVisit(G, v)
```

Procedimiento dfsVisit(G, v)

```
1  inicio
2      tiempo ← tiempo + 1 ; /* se empieza a explorar v                */
3      v.cc ← contCC ;
4      v.ti ← tiempo ; /* tiempo inicial                               */
5      v.color ← GRIS ;
6      para cada (u ∈ G.adyacentes[v]) hacer
7          si (u.color = BLANCO) entonces
8              u.pred ← v ;
9              dfsVisit(G, u)
10     v.color ← NEGRO ; /* se termina de explorar v                   */
11     tiempo ← tiempo + 1 ;
12     v.tf ← tiempo ; /* tiempo final                                 */
```


Detectando componentes conexas con DFS, cont.

Prueba

Se quiere probar que en una llamada de `dfsVisit` desde `componenteConexaDFS`, se visitan todos los vértices de una componente conexa:

- Sea v el primer vértice de una componente conexa que entra en `dfsVisit`.
- Como para entrar en `dfsVisit` v tiene que ser blanco, entonces todos los vértices de la componente conexa tienen que ser blancos.
- Todos los vértices van a ser descendientes de v en el bosque (Teorema del camino blanco).
- Entonces los vértices van a ser visitados con la llamada recursiva de `dfsVisit`.
- Entonces a todos se les va asignar la misma componente conexa (línea 3 `dfsVisit`).

Detectando componentes conexas con DFS, cont.

Continuación de la prueba

Se quiere probar que todos los vértices de una componente conexa, se visitan en una llamada de `dfsVisit` desde `componenteConexaDFS`:

- Si dos vértices son visitados en la misma llamada de `dfsVisit` hubo un camino desde un vértice hasta otro porque entran en `dfsVisit` recursivo (línea 8 de `dfsVisit`) por medio de la lista de adyacencias.
- Como hay un camino de un vértice hasta otro, entonces perteneces a la misma componente conexa.

□

Sobre la estructura de conjuntos disjuntos

Sobre la estructura de conjuntos disjuntos

- Se representa a una colección dinámica de n conjuntos disjuntos $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$.
- Dado un elemento x, se denota como a S_x al conjunto que lo contiene.
- Cada conjunto S_i tiene un elemento representativo $rep[S_i]$.
- Para determinar si dos elementos x y y pertenecen a mismo conjunto, se verifica si $rep[S_x] = rep[S_y]$.

Operaciones sobre la estructura de conjuntos disjuntos

- MAKE-SET(X):
  - Crea un nuevo conjunto conteniendo al elemento x.
  - x no debe estar contenido en ningún otro conjunto de $\mathcal{S}$
  - x es el elemento representativo del conjunto actualmente
- FIND-SET(X): Retorna el elemento representativo del conjunto S_x que contiene a x, esto es $rep[S_x]$
- UNION(X, Y): Se reemplaza a S_x y S_y por $S_x \cup S_y$ en $\mathcal{S}$. Se actualiza el elemento representativo del nuevo conjunto.

Obteniendo componentes conexas

Componentes conexas como conjuntos disjuntos

- Se quiere obtener las componentes conexas de un grafo no dirigido.
- Se utilizan las operaciones de la estructura conjuntos disjuntos para obtener las componentes conexas.
- Los conjuntos disjuntos $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ son las n componentes conexas de un grafo no dirigido.
- Cada componente conexa es un conjunto disjunto S_i.
- Se verifica si $rep[S_u] = rep[S_v]$, para saber si los vértices u y v están en la misma componente conexa.

Determinando las componentes conexas

Procedimiento componenteConexaCD($G = (V, E)$) G es no dirigido

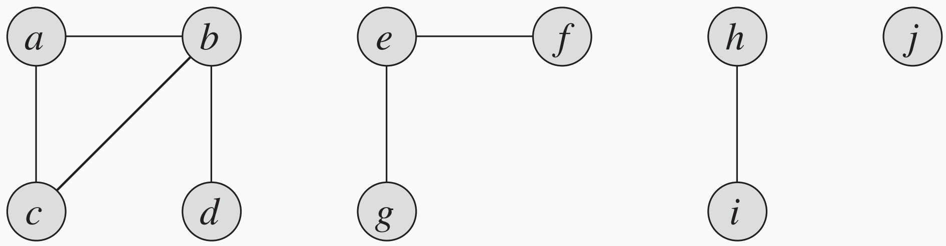
```
inicio
    para cada v ∈ V hacer
        make-set(v)
    para cada (u, v) ∈ E hacer
        si find-set(u) ≠ find-set(v) entonces
            union(u, v)
```


Verificando a las componentes conexas

Función estanEnLaMismaComponenteConexa(v, u)

```
inicio
    si find-set(u) = find-set(v) entonces
        devolver True
    en otro caso
        devolver False
```

Ejemplo de obtención de las componentes conexas



(a) Grafo G de entrada con cuatro componentes conexas

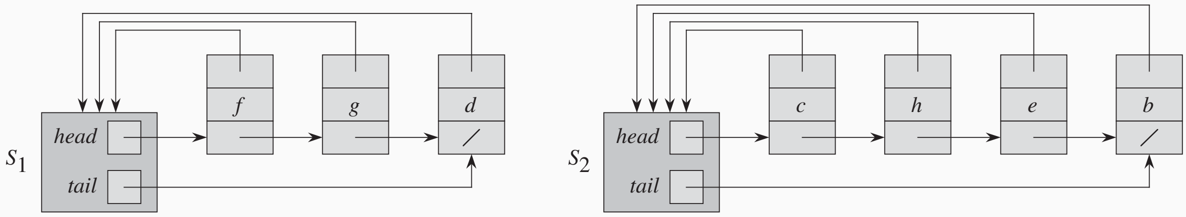
Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

(b) Ejecución de componenteConexaCD(G).

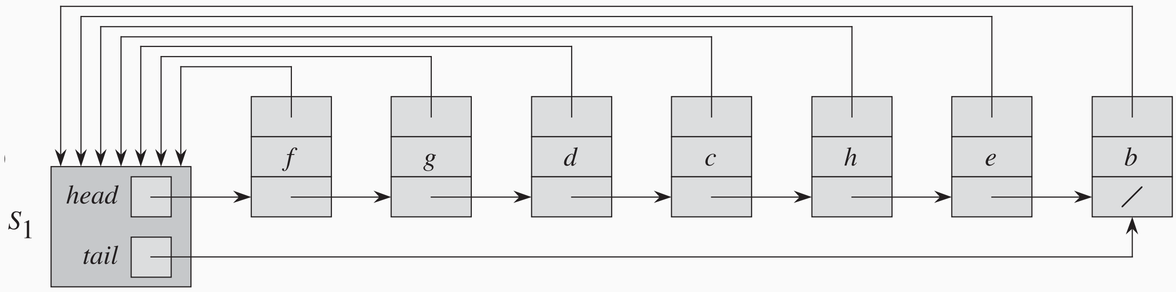
Figura 1: Componentes conexas como conjuntos disjuntos. Fuente [1].

Representación como listas enlazadas

Ejemplo de implementación como listas enlazadas



(a) Dos conjuntos representados como listas enlazadas



(b) Unión de dos conjuntos de (a)

Figura 2: Conjuntos disjuntos como listas enlazadas. Fuente [1].

Análisis de las operaciones con listas enlazadas

- $\textsc{make-set}(\textsc{x})$ es $O(1)$ en el peor caso.
- $\textsc{find-set}(\textsc{x})$ es $O(1)$ en el peor caso.
- $\textsc{union}(\textsc{x}, \textsc{y})$ es $O(n)$ en el peor caso y $\Theta(n)$ en el caso amortizado.

Heurística de la unión pesada

Heurística de la unión pesada

Al ejecutar UNION(X, Y) siempre se concatena la lista más corta (con menos elementos) con la más grande (con más elementos). Esto hace que el tiempo de UNION en el peor caso sea $\Omega(n)$.

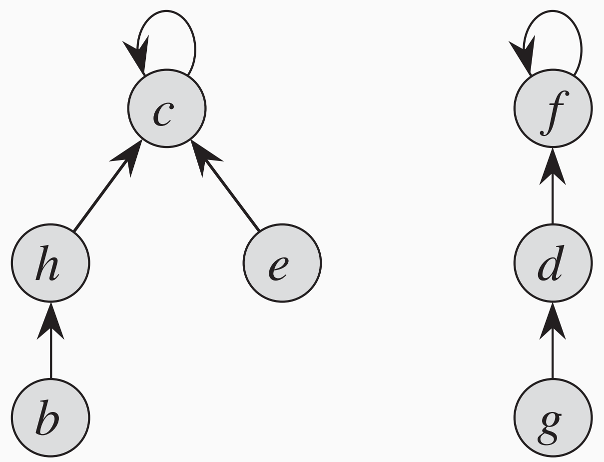
Heurística de la unión pesada, cont.

Teorema 1

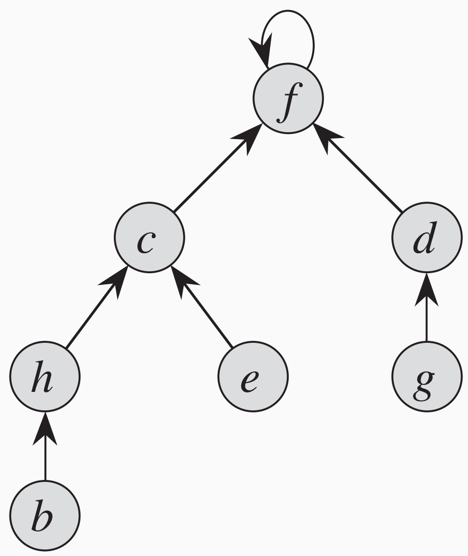
Usando la representación con listas enlazas y la heurística de la unión pesada, una secuencia de m operaciones de MAKE-SET, FIND-SET y UNION, de las cuales n operaciones son MAKE-SET, se ejecutan en un tiempo de $O(m + n \lg n)$.

Representación como árboles

Ejemplo de implementación como árboles



(a) Dos conjuntos representados como árboles



(b) Unión de dos conjuntos de (a)

Figura 3: Conjuntos disjuntos como árboles. Fuente [1].

Heurística para mejorar las operaciones con árboles

Heurística de la unión por rank

Cada nodo del árbol lleva su valor de *rank* que es una cota superior de su altura. Al ejecutar UNION(X, Y) siempre se concatena el árbol con menor *rank* con el de *rank* más grande.

Heurística de la compresión

Cada vez que se ejecute FIND-SET(X), el nodo x y todos los nodos en el camino hasta x, van apuntar a la raíz. El *rank* de los nodos del árbol no cambia.

Ejemplo de compresión de caminos

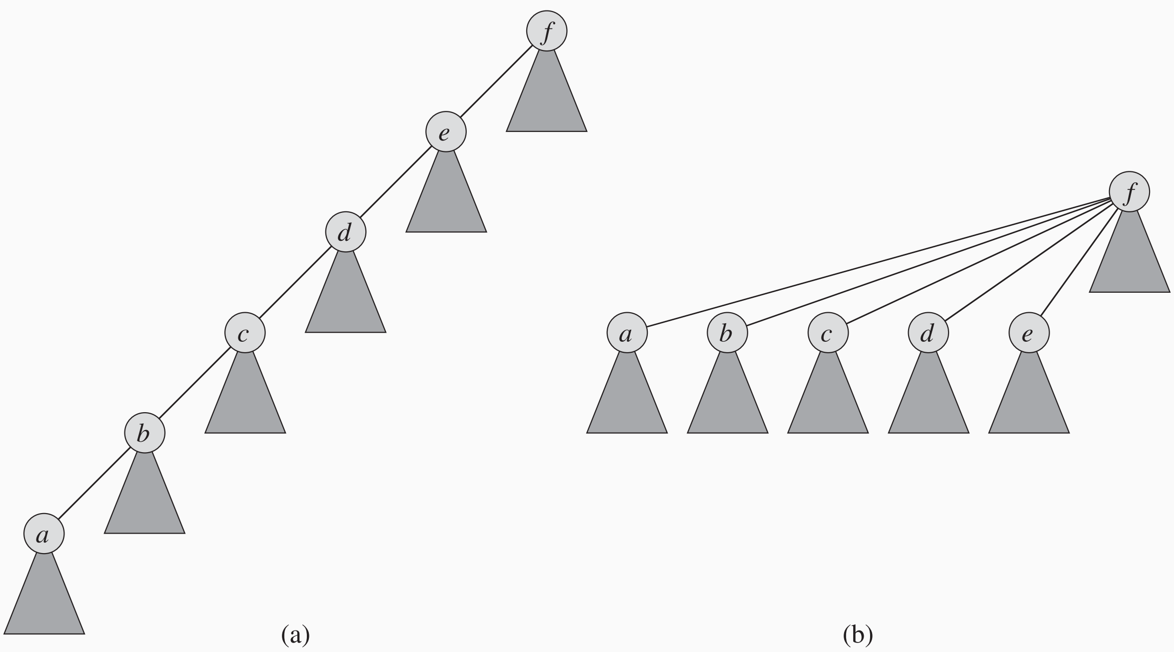


Figura 4: Compresión de caminos después de la operación FIND-SET. En (a) un árbol antes de ejecutar FIND-SET(a). En (b) el árbol después de ejecutar FIND-SET(a). Fuente [1].

Operaciones de conjuntos disjuntos con árboles

Procedimiento make-set(x)

inicio
  $x.p \leftarrow x$; $x.rank \leftarrow 0$

Procedimiento union(x, y)

inicio
  `link(find-set(`x`), find-set(`y`))`

Procedimiento link(x, y)

inicio
  si $x.rank > y.rank$ **entonces**
    $y.p \leftarrow x$
  en otro caso
    $x.p \leftarrow y$ **si** $x.rank = y.rank$ **entonces** $y.rank \leftarrow y.rank + 1$;

Operaciones de conjuntos disjuntos con árboles, cont.

Función find-set(x)

inicio
  si $x \neq x.p$ **entonces**
    $x.p \leftarrow$ `find-set(`$x.p$`)`
  devolver $x.p$

Análisis de las operaciones con árboles

Sin las heurísticas *rank* y compresión se tiene que:

- MAKE-SET(X) es $O(1)$ en el peor caso.
- FIND-SET(X) es $O(altura) = \Theta(n)$ en el peor caso.
- UNION(X, Y) es $\Theta(n)$ en el peor caso.

Teorema 2

Usando la representación con árboles y la heurísticas de *rank* y compresión, una secuencia de m operaciones de MAKE-SET, FIND-SET y UNION, de las cuales n operaciones son MAKE-SET, se ejecutan en un tiempo en el peor caso de $O(m\alpha(n))$ donde $\alpha(n) \leq 4$.

Referencias

[1] T. Cormen, C. Leirserson, R. Rivest, and C. Stein.
Introduction to Algorithms.
McGraw Hill, 3ra edition, 2009.