Estructuras de datos: Conjuntos disjuntos

Algoritmos

Dep. de Computación - Fac. de Informática
Universidad de A Coruña

Santiago Jorge
sjorge@udc.es

Referencias bibliográficas

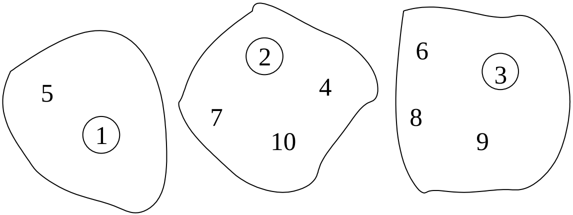
- G. Brassard y T. Bratley. Estructura de datos. En *Fundamentos de algoritmia*, capítulo 5, páginas 167–210. Prentice Hall, 1997.
- M. A. Weiss. El TDA conjunto ajeno. En *Estructuras de datos y algoritmos*, capítulo 8, páginas 271–292. Addison-Wesley Iberoamericana, 1995.
- A. V. Aho, J. E. Hopcroft y J. D. Ullman. Métodos avanzados de representación de conjuntos. En *Estructuras de datos y algoritmos*, capítulo 5, páginas 157–199. Addison-Wesley Iberoamericana, 1988.
- U. Manber. Data Structures. En *Introduction to Algorithms. A Creative Approach*, capítulo 4, páginas 61–90. Addison-Wesley, 1989.

Índice

Índice

Representación de conjuntos disjuntos

- Todos los elementos se numeran de 1 a *n*.
- Cada subconjunto tomará su nombre de uno de sus elementos, su representante, p. ej. el valor más pequeño.
- Mantenemos en un vector el nombre del subconjunto disjunto de cada elemento



	1	2	3	4	5	6	7	8	9	10
C	1	2	3	2	1	3	2	3	3	2

Operaciones válidas

- La representación inicial es una colección de n conjuntos, C_i.
    - Cada conjunto tiene un elemento diferente, $C_i \cap C_j = \emptyset$
    - Así, al principio se tiene $C_i = \{i\}$.
- Hay dos operaciones válidas.
    - La búsqueda devuelve el nombre del conjunto de un elemento dado.
    - La unión combina dos subconjuntos que contienen a y b en un subconjunto nuevo, destruyéndose los originales.

Pseudocódigo (i)

```
tipo
  Elemento = entero;
  Conj = entero;
  ConjDisj = vector [1..N] de entero

función Buscar1 (C, x) : Conj
  devolver C[x]
fin función
```

- La *búsqueda* es una simple consulta $O(1)$.
  - El nombre del conjunto devuelto por *búsqueda* es arbitrario.
  - Todo lo que importa es que *búsqueda(x)=búsqueda(y)* si y sólo si x e y están en el mismo conjunto.

Pseudocódigo (ii)

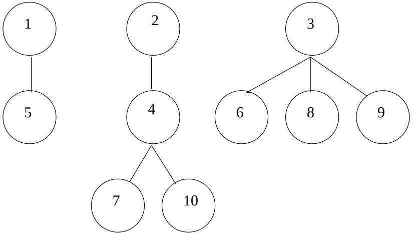
```
procedimiento Unir1 (C, a, b)
  i := min (C[a], C[b]);
  j := max (C[a], C[b]);
  para k := 1 hasta N hacer
    si C[k] = j entonces C[k] := i
  fin para
fin procedimiento
```

- La *unión* toma $O(n)$. No importa, en lo que concierne a corrección, qué conjunto retiene su nombre.
- Una secuencia de $n-1$ uniones (la máxima, ya que entonces todo estaría en un conjunto) tomaría $O(n^2)$.
- La combinación de m búsquedas y $n-1$ uniones toma $O(m+n^2)$.

Índice

Segundo enfoque

- Se utiliza un árbol para caracterizar cada subconjunto.
  - La raíz nombra al subconjunto.
  - La representación de los árboles es fácil porque la única información necesaria es un apuntador al padre.
  - Cada entrada *p[i]* en el vector contiene el padre del elemento *i*.
    - Si *i* es una raíz, entonces *p[i]=i*



	1	2	3	4	5	6	7	8	9	10
C	1	2	3	2	1	3	4	3	3	4

Pseudocódigo (i)

```
función Buscar2 (C, x) : Conj
  r := x;
  mientras C[r] <> r hacer
    r := C[r]
  fin mientras;
  devolver r
fin función
```

- Una búsqueda sobre el elemento x se efectúa devolviendo la raíz del árbol que contiene x.
- La *búsqueda* de un elemento x es proporcional a la profundidad del nodo con x.
  - En el peor caso es $O(n)$

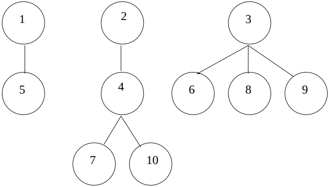
Pseudocódigo (ii)

```
procedimiento Unir2 (C, raíz1, raíz2)
           { supone que raíz1 y raíz2 son raíces }
  si raíz1 < raíz2 entonces C[raíz2] := raíz1
  sino C[raíz1] := raíz2
fin procedimiento
```

- La unión de dos conjuntos se efectúa combinando ambos árboles: apuntamos la raíz de un árbol a la del otro.
- La unión toma $O(1)$.
- La combinación de m búsquedas y $n-1$ uniones toma $O(m \cdot n)$.

Unión por alturas

- Las uniones anteriores se efectuaban de modo arbitrario.
- Una mejora sencilla es realizar las uniones haciendo del árbol menos profundo un subárbol del árbol más profundo.
  - La altura se incrementa sólo cuando se unen dos árboles de igual altura.



	1	2	3	4	5	6	7	8	9	10
C	1	2	3	2	1	3	4	3	3	4

	1	2	3	4	5	6	7	8	9	10
A	1	2	1							

Pseudocódigo

```
procedimiento Unir3 (C, A, raíz1, raíz2)
            { supone que raíz1 y raíz2 son raíces }
  si A[raíz1] = A[raíz2] entonces
    A[raíz1] := A[raíz1] + 1;
    C[raíz2] := raíz1
  sino si A[raíz1] > A[raíz2] entonces C[raíz2] := raíz1
  sino C[raíz1] := raíz2
fin procedimiento
```

- La profundidad de cualquier nodo nunca es mayor que $\log_2(n)$.
  - Todo nodo está inicialmente a la profundidad 0.
  - Cuando su profundidad se incrementa como resultado de una unión, se coloca en un árbol al menos el doble de grande.
    - Así, su profundidad se puede incrementar a lo más, $\log_2(n)$ veces.
- El tiempo de ejecución de una búsqueda es $O(\log(n))$.
- Combinando m búsquedas y $n-1$ uniones, $O(m \cdot \log(n) + n)$.

Compresión de caminos

- La compresión de caminos se ejecuta durante búsqueda.
  - Durante la búsqueda de un dato x, todo nodo en el camino de x a la raíz cambia su padre por la raíz.
  - Es independiente del modo en que se efectúen las uniones.

```
función Buscar3 (C, x) : Conj
  r := x;
  mientras C[r] <> r hacer
    r := C[r]
  fin mientras;
  i := x;
  mientras i <> r hacer
    j := C[i]; C[i]:= r; i := j
  fin mientras;
  devolver r
fin función
```

Rangos

- La compresión de caminos no es totalmente compatible con la unión por alturas.
  - Al buscar, la altura de un árbol puede cambiar.
- Las alturas almacenadas para cada árbol se convierten en alturas estimadas (rangos).
  - Pero la unión por rangos es tan eficiente, en teoría, como la unión por alturas.

Ejemplo de compresión de caminos