SUBCONJUNTOS DISJUNTOS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Subconjuntos disjuntos

Operações (DS = disjoint set):

- void makeset(DS ds, E x); // realizada uma única vez para cada x
- set find(DS ds, E x);
- void union(DS ds, E x, E y);

Exemplo: $S = \{1, 2, 3, 4, 5, 6\}$

- $makeset(ds, 1);\ makeset(ds, 2);\ makeset(ds, 3);$
  $makeset(ds, 4);\ makeset(ds, 5);\ makeset(ds, 6);$
- $union(ds, 1, 4);\ union(ds, 5, 2);$
  $union(ds, 4, 5);\ union(ds, 3, 6);$
- $find(ds, 2);\ find(ds, 1);\ find(ds, 3);$

Conveniente: elemento representativo

Implementações: quick-find e quick-union

Subconjuntos disjuntos

Implementação baseada em quick-find

- Array indexado pelos elementos, armazenando representativo
- Cada subconjunto é implementado como uma lista ligada
- *makeset*: atribuir x como representativo de x e iniciar lista – $\Theta(1)$
- *find*: retornar representativo de x – $\Theta(1)$
- *union*: concatenar duas listas e atualizar representativos – $\Theta(n)$

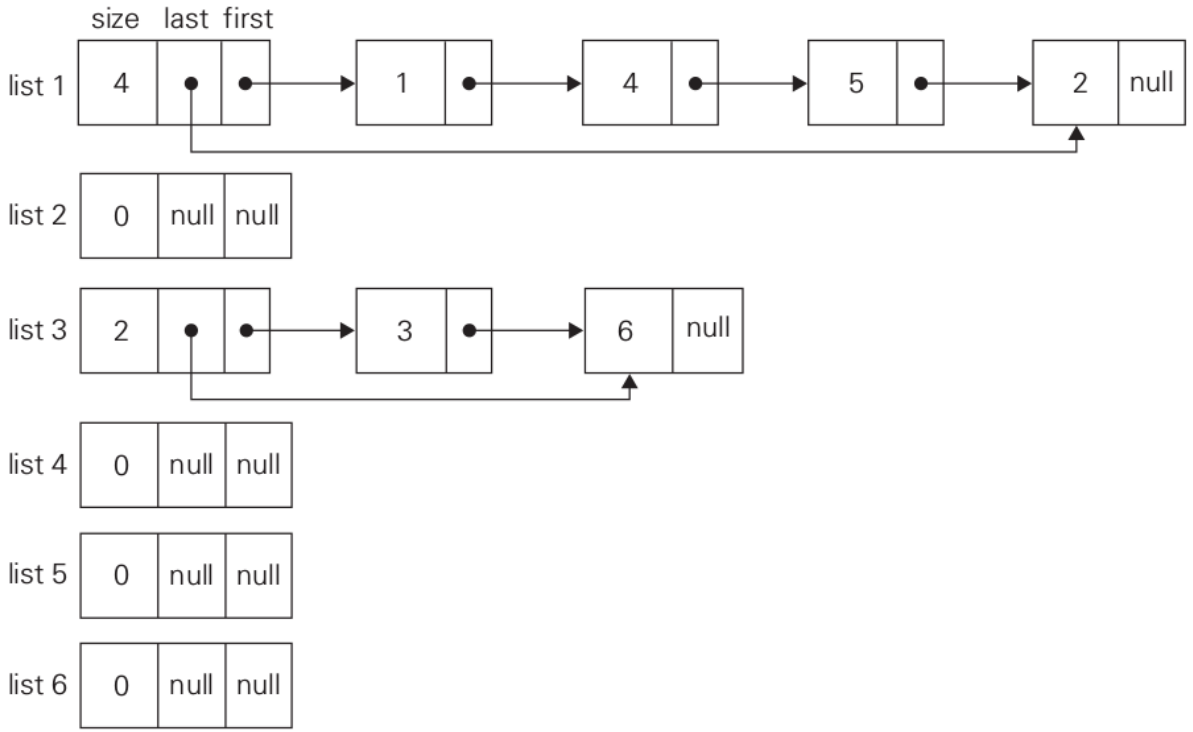
Otimização (union by size): concatenar menor na maior lista

Sequência de até $n-1$ unions e m finds: $O(n \log n + m)$

Subconjuntos disjuntos

Implementação baseada em quick-find

subset representatives	
element index	representative
1	1
2	1
3	3
4	1
5	1
6	3



1

[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Subconjuntos disjuntos

Implementação baseada em quick-union

Cada subconjunto como uma árvore (raiz: elemento representativo)

- Filhos apontam para os pais (parent pointer tree)
- Array associando elementos aos seus respectivos nós

Subconjuntos disjuntos

Implementação baseada em quick-union

- *makeset*: criar árvore unitária e associar elemento ao nó – $\Theta(1)$
- *find*: percorrer de x até a raiz – $\Theta(n)$
- *union*: raiz de y aponta para raiz de x – $\Theta(n)$

Otimização (union by size/rank): concatenar menor na maior árvore

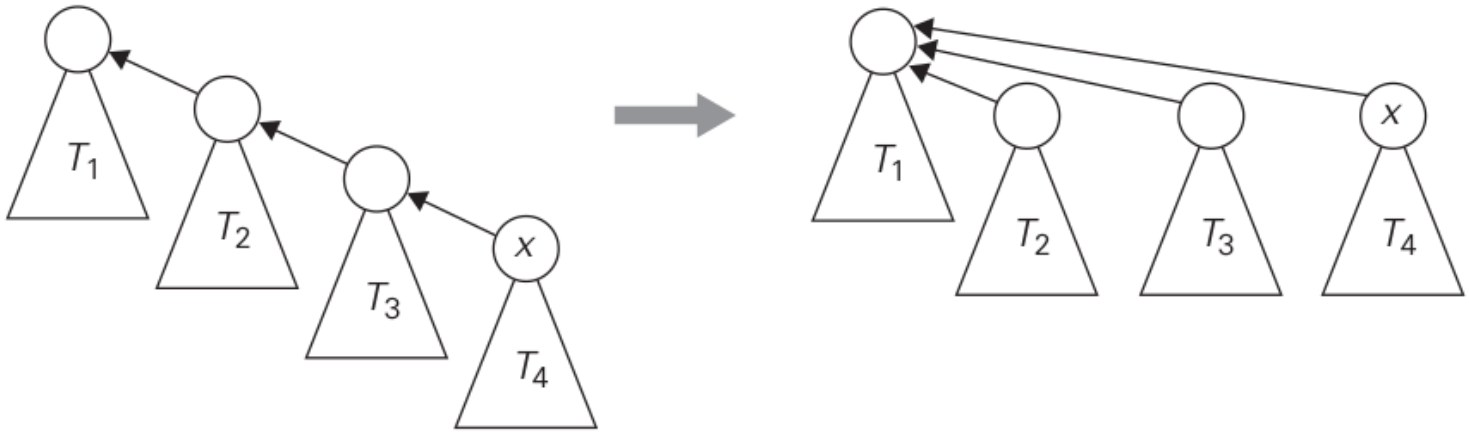
- Size: número de nós (armazenar quantidade descendentes)
- Rank: altura (armazenar altura sub-árvores)

Sequência de até $n-1$ unions e m finds: $O(n + m \log n)$

Subconjuntos disjuntos

Implementação baseada em quick-union

- Otimização adicional: compressão de caminhos durante find
- $n - 1$ unions e m finds: ligeiramente pior do que linear



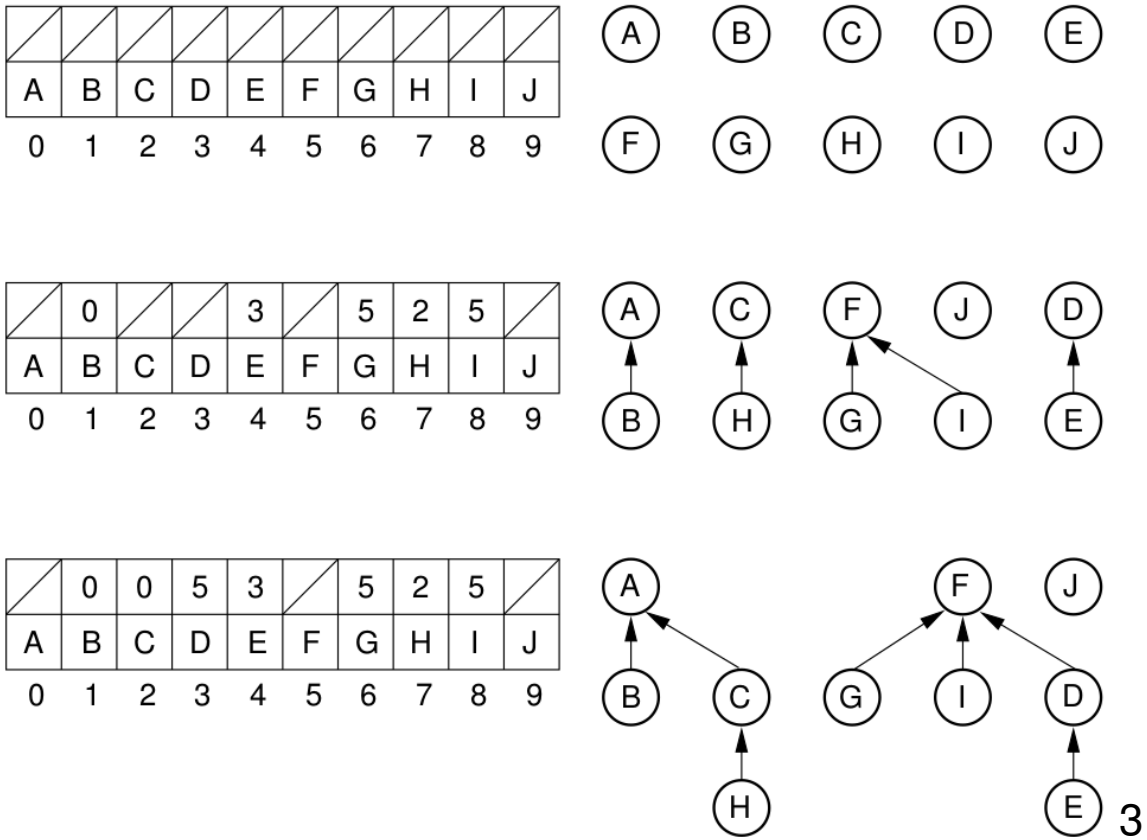
2

[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Implementação de quick-union baseada em array

Operações

- $union(ds, A, B), union(ds, C, H), union(ds, G, F), union(ds, D, E), union(ds, I, F)$
- $union(ds, H, A), union(ds, E, G)$



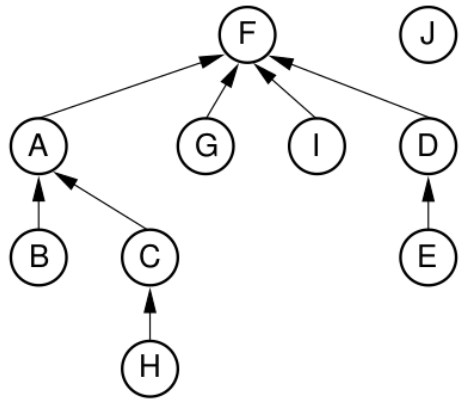
[3]

[3] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Implementação de quick-union baseada em array

Operação: *union*(*ds*, *H*, *E*)

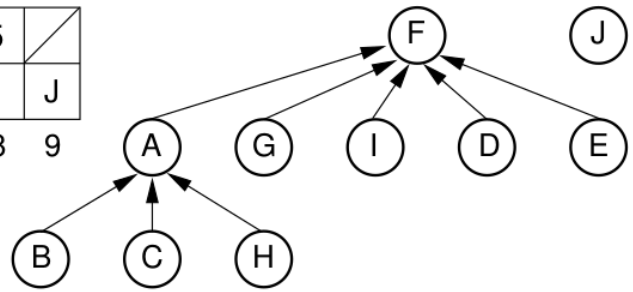
5	0	0	5	3	/	5	2	5	/
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



4

Com compressão de caminho: *union*(*ds*, *H*, *E*)

5	0	0	5	5	/	5	0	5	/
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



5

[4] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

[5] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Implementação de quick-union baseada em array

Algoritmo: int find(A [0..n-1], int curr)

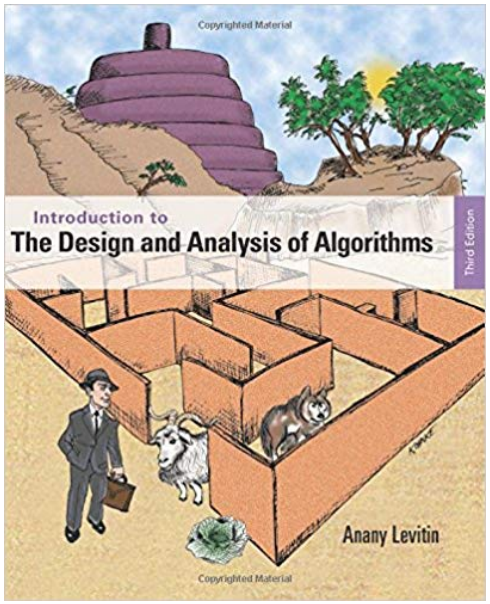
```
1  if A[curr] = NULL then return curr;
2  A[curr] ← find(A, A[curr]);
3  return A[curr];
```

Algoritmo: void union(A [0..n-1], int a, int b)

```
1  int root1 ← find(A, a);
2  int root2 ← find(A, b);
3  if root1 ≠ root2 then A[root2] ← root1;
```

Agenda

Bibliografia + leitura recomendada



Capítulo 9 (pp. 327–331)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

Capítulo 6 (pp. 199–206)
Clifford Shaffer.
Data Structures and Algorithm Analysis.
Dover, 2013.

SUBCONJUNTOS DISJUNTOS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil