ESTRUTURAS DE DADOS PARA CONJUNTOS DISJUNTOS (Cormen, §21.1-2)

Em alguns algoritmos, como o algoritmo de árvore geradora mínima de Kruskal e um algoritmo de escalonamento de tarefas, é necessário manter certos elementos agrupados em conjuntos disjuntos (isto é, um elemento nunca pertence a dois conjuntos simultaneamente), e, em cada momento, as operações possíveis são apenas 3: (1) criar um conjunto unitário (com elemento que não pertença a outros conjuntos), (2) unir dois conjuntos, e (3) consultar a que conjunto um certo elemento pertence.

Um fato importante é o de que, nas aplicações citadas, a operação de consulta de conjunto é utilizada apenas para verificar se dois dados elementos pertencem ou não ao mesmo conjunto. Nesse caso, a maneira exata de se identificar um conjunto não é importante, desde que se possa discernir entre elementos de conjuntos diferentes e elementos de um mesmo conjunto. Assim, na prática, o que se faz é identificar um conjunto por meio de um REPRESENTANTE, que é simplesmente um dos elementos do conjunto, MAS QUE, PARA CADA CONJUNTO, NUNCA MUDA.

Assim sendo, as operações em questão são precisamente as seguintes:

* "Criar Conjunto": essa operação recebe como entrada um elemento "x", e cria
  ---------------- um conjunto cujo único elemento é "x". Naturalmente, o representante do conjunto em questão é necessariamente o elemento "x". Além disso, como os conjuntos manipulados têm que ser disjuntos, essa operação não pode ser realizada mais de uma vez sobre o mesmo elemento.

* "Unir Conjuntos": essa operação recebe dois elementos "a" e "b", que têm que
  ---------------- pertencer respectivamente a conjuntos A e B, sendo A != B. A operação então cria um conjunto C, cujos elementos são exatamente aqueles de A e de B, e destrói os conjuntos A e B. Em princípio, o representante do novo conjunto pode ser qualquer um dos seus elementos; na prática, ele é frequentemente escolhido entre o representante de A e o representante de B.

* "Consultar Conjunto": essa operação recebe um elemento "x", que tem que
  -------------------- pertencer a um conjunto C, e então retorna o representante de C.

Na prática, um algoritmo qualquer realiza uma sequência de "m" chamadas às operações acima, "n" das quais sendo chamadas à operação "criar conjunto", e "m - n" sendo chamadas às operações "unir conjuntos" e "consultar conjunto". Naturalmente, temos que n <= m. Além disso, como são criados apenas "n" conjuntos, o número de chamadas à operação "unir conjuntos" é no máximo "n - 1".

1. EXERCÍCIO: você vislumbra alguma implementação eficiente para o tipo abstrato
   --------- de dados descrito acima? Se sim, qual é a complexidade assintótica (em termos de "m" e "n") de uma sequência de operações nessa implementação?

IMPLEMENTAÇÃO POR LISTAS ENCADEADAS

Uma maneira simples de se implementar o tipo abstrato de dados para conjuntos disjuntos é representar cada conjunto por meio de uma lista encadeada. Nesse caso, parte da operação de união consiste simplesmente em concatenar as listas dos conjuntos dos elementos de entrada, o que pode ser feito em tempo constante (assim como a operação de criação de um conjunto). Entretanto, para que a operação de consulta execute rapidamente, é conveniente que cada elemento "x" possua uma referência essencialmente direta para o representante do conjunto de "x"; nesse caso, essas referências precisam ser atualizadas durante a operação de união, que então deixa de executar em tempo constante.

2. EXERCÍCIO: mostre que, se uma implementação de conjuntos disjuntos nos moldes
   --------- descritos acima for tal que, na operação de união, as referências
   atualizadas são sempre as dos elementos do conjunto do segundo operando,
   então existem sequências de O(n) operações que executam em tempo $\theta(n^2)$
   (naturalmente, o mesmo acontecendo no caso de sempre serem atualizadas as
   referências dos elementos do conjunto do primeiro operando).

O fato observado no exercício acima pode ser facilmente contornado: basta que cada conjunto armazene também o número de elementos que possui, e que, numa operação de união, sejam sempre atualizadas as referências dos elementos do menor conjunto. Como demonstrado na sequência, isso limita o tempo de execução de "m" operações em O(m + n*lg n), sendo "n" o número de operações de criação de conjuntos.

As observações acima levam diretamente aos algoritmos abaixo (ATENÇÃO: abaixo, a notação "P[x]" não necessariamente significa "o x-ésimo elemento do vetor P"; ao invés disso, pode significar "o atributo P da entidade x"; essa notação é utilizada no Cormen, 2ª edição, e, apesar de ambígua do ponto de vista da implementação, é conveniente quando familiar):

```
================================================================================
Algoritmo: criar_conjunto
Entrada:   um elemento "x".
Saída:     nenhuma.
--------------------------------------------------------------------------------
1. L := nova lista encadeada vazia
2. insira x em L
3. tam[L] := 1
4. lista[x] := L
================================================================================
```

```
================================================================================
Algoritmo: consultar_conjunto
Entrada:   um elemento "x".
Saída:     um elemento.
--------------------------------------------------------------------------------
1. Retorne o primeiro elemento de lista[x].
================================================================================
```

```
================================================================================
Algoritmo: unir_conjuntos
Entrada:   elementos "x" e "y".
Saída:     nenhuma.
--------------------------------------------------------------------------------
01. A := lista[x], B := lista[y]
02. SE tam[A] >= tam[B] ENTÃO LG := A, LP := B
03. |                   SENÃO LG := B, LP := A
04. FIM_SE
05. novo_tam := tam[LG] + tam[LP]
06. PARA cada elemento z de LP
07. | insira z no final de LG
08. | lista[z] := LG
09. FIM_PARA
10. tam[LG] := novo_tam
11. destrua LP.
================================================================================
```

3. LEMA: se, partindo-se de uma estrutura vazia, são realizadas "m" chamadas às
   ---- funções "criar_conjunto", "unir_conjuntos" e "consultar_conjunto",
   sendo "n" destas à função "criar_conjunto", então o tempo total da execução
   das "m" chamadas é O(m + n*lg n).

==========
PROVA:

Observe primeiramente que cada chamada a "criar_conjunto" ou a "consultar_conjunto" executa em tempo constante, de forma que todas as chamadas a essas funções executam num tempo total de O(m). Além disso, cada chamada a "unir_conjuntos" executa em tempo O(k), sendo "k" o número de vezes que a instrução "lista[z] := LG" é executada na chamada em questão.

Nós mostraremos agora que, para cada elemento "x", a instrução em questão é executada no máximo ⌊lg n⌋ vezes. De fato, seja um elemento "x", isto é, um objeto que foi fornecido como argumento para uma chamada de "criar_conjunto". Claramente, se a instrução "lista[x] := LG" é executada "k" vezes, então é porque, em "k" operações de união, um conjunto contendo "x" era o menor dos dois conjuntos que eram unidos. Logo, após as "k" execuções da instrução em questão, "x" pertence a um conjunto que possui pelo menos 2^k elementos. Agora, como ocorreram apenas "n" chamadas à função "criar_conjunto", então "x" sempre pertence a um conjunto com no máximo "n" elementos, ou seja, 2^k <= n, o que implica que k <= ⌊lg n⌋, como desejávamos.

Logo, como o número de elementos "z" sobre os quais a instrução "lista[z] := LG" é executada é no máximo "n", então as chamadas à função "unir_conjuntos" executam, ao todo, em tempo O(n*⌊lg n⌋) = O(n*lg n), do que segue que o tempo total das "m" chamadas é O(m + n*lg n), CQD.

==========

4. EXERCÍCIO: implemente os algoritmos acima.

 LEITURA ADICIONAL

Há uma estrutura de dados para conjuntos disjuntos que é simples e mais eficiente que essa apresentada acima (a prova do tempo de execução, porém, é bem mais complicada). Consulte o Cormen, §21.3.