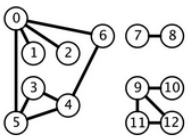
A estrutura union-find

Esta página discute o tipo abstrato de dados (= *abstract data type* = *ADT*) conhecido como union-find. O conceito é introduzido no contexto de um algoritmo para calcular o número de componentes de um grafo. O union-find é usado por implementações eficientes de vários algoritmos interessantes, como o de Kruskal (para o problema da MST) e o de Karger (para cortes mínimos).

Esta página é inspirada em partes do capítulo 21 do livro de CLRS.

Cálculo das componentes

Como calcular o número de componentes de um grafo? Como determinar, eficientemente, se dois dados vértices do grafo estão na mesma componente?



Para começar, convém que cada componente do grafo tenha um "nome" que permita distinguir uma componente da outra. Para isso, usaremos a seguinte ideia. Imagine que cada componente é uma empresa e os vértices da componente são os funcionários da empresa. Um dos funcionários é o *diretor* da empresa. Assim, o diretor servirá para identificar a empresa.

Usaremos o tipo abstrato de dados *union-find* para contar as componentes do grafo. Esse tipo de dados dispõe de duas operações:

- a operação FIND recebe um vértice v e devolve o diretor da componente a que v pertence;
- a operação UNION recebe os diretores, digamos r e s, de duas componentes distintas, faz a fusão das duas componentes, e designa r ou s como diretor da nova componente.

Dado um grafo G com n vértices, as operações do union-find podem ser usadas como segue para calcular o número de componentes:

```
COMPONENTES (G, n)
1  c := n
2  INITIALIZE ( )
3  para cada aresta uv de G
4     r := FIND (u)
5     s := FIND (v)
6     se r ≠ s
```

```
7          UNION (r, s)
8          c := c − 1
9   devolva c
```

A implementação das operações FIND e UNION pode precisar de alguma estrutura de dados auxiliar; essa estrutura é prepara e inicializada pela operação INITIALIZE.

As seções seguintes mostrarão diversas implementações das operações UNION e FIND. O desafio é obter uma implementação em que ambas as operações consumam tempo constante, ou seja, tempo independente de n.

Exercícios 1

1. Como usar o algoritmo COMPONENTES para decidir se um grafo é conexo?

Primeira implementação

Nossa primeira implementação do tipo abstrato de dados union-find é muito simples. Os diretores das componentes ficam armazenados num vetor dir indexado pelos vértices do grafo: para cada vértice v, $dir[v]$ é o diretor da componente que contém v. A implementação de FIND é óbvia:

```
FIND-1 (v)
1   devolve dir[v]
```

Um vértice v é o diretor de alguma componente se e somente se $dir[v] = v$. No início dos trabalhos, cada componente tem um só vértice, que é também o diretor da componente:

```
INITIALIZE-1 ( )
1   para cada vértice v
2       dir[v] := v
```

A operação UNION recebe dois diretores e faz a fusão de suas componentes. Para isso, basta informar todos os vértices de uma das componentes que eles têm um novo diretor:

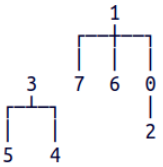
```
UNION-1 (r, s)  ▷ r ≠ s
1   para cada vértice v
2       se dir[v] = r
3           dir[v] := s
```

A operação FIND-1 é muito rápida. Já a operação UNION-1 é muito lenta: ela que consome $\Omega(n)$ unidades de tempo no pior caso, sendo n o número de vértices do grafo.

Exercícios 2

1. Mostre um exemplo completo em que UNION-1 consome tempo proporcional a n.

Segunda implementação



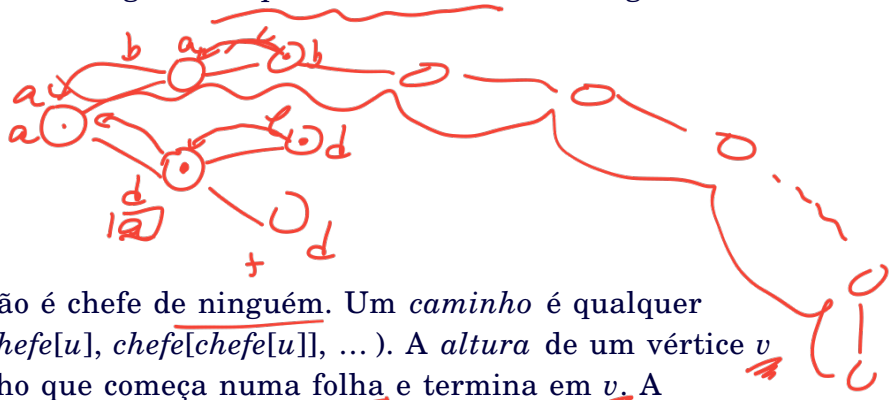
Para tornar o operação UNION mais rápida, vamos introduzir a ideia de "superior imediato", ou "chefe", na hierarquia corporativa. Imagine que cada vértice de uma componente tem um chefe, que é um outro vértice da mesma componente. Cada chefe, por sua vez, tem um chefe, e assim por diante, até chegar a um vértice que é chefe dele mesmo. Essa ideia precisa ser implementada de tal modo que um único vértice de cada componente seja chefe dele mesmo; esse vértice faz o papel de diretor da componente.

Os chefes ficam armazenados num vetor *chefe* de modo que *chefe*[*v*] é o chefe do vértice *v*. O vetor *dir* da implementação anterior desaparece, e um vértice *v* é considerado diretor se e somente se *chefe*[*v*] = *v*. A inicialização do vetor *chefe* é simples:

```
INITIALIZE-2 ( )
1   para cada vértice v
2       chefe[v] := v
```

Para implementar a operação FIND, basta seguir a sequência de chefes até chegar ao diretor:

```
FIND-2 (v)
1   enquanto chefe[v] ≠ v
2       v := chefe[v]
3   devolva v
```



Uma *folha* é qualquer vértice que não é chefe de ninguém. Um *caminho* é qualquer sequência de vértices da forma (*u*, *chefe*[*u*], *chefe*[*chefe*[*u*]], ...). A *altura* de um vértice *v* é comprimento do mais longo caminho que começa numa folha e termina em *v*. A *profundidade* de um vértice *v* é o comprimento do único caminho que começa em *v* e termina num diretor.

O consumo de tempo de FIND-2 é proporcional à profundidade de *v* e portanto, no pior caso, proporcional ao máximo das alturas dos diretores. Infelizmente, a altura de um diretor pode chegar a $n-1$. Portanto, o algoritmo FIND-2 é lento.

Em compensação, a implementação de UNION é muito rápida:

```
UNION-2 (r, s)  ▷ r ≠ s
1   chefe[r] := s
```

Exercícios 3

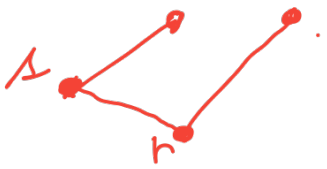
1. ★ Suponha que *next* é um vetor que leva cada vértice do grafo num outro vértice da mesma componente. Esse vetor pode ser um vetor de chefes?
2. Mostre um exemplo completo em que a operação FIND-2 consome tempo proporcional a n.
3. ★ Escreva uma versão recursiva da operação FIND-2.

Terceira implementação: union-by-rank

Para fugir do mau desempenho de FIND-2, é preciso limitar a altura dos diretores. Um truque simples garante esse efeito: a operação UNION deve fazer com que o diretor da *maior* das duas componentes passe a ser o diretor da fusão das duas componentes. Essa heurística é conhecida como *union-by-rank*.

Para implementar a heurística, é preciso manter na estrutura de dados a informação sobre a altura de cada vértice. As alturas dos vértice serão mantidas em um vetor *alt* de modo que *alt*[*v*] seja a altura do vértice *v*.

```
UNION-3 (r, s)  ▷ union-by-rank; r ≠ s
1   se alt[r] > alt[s]
2       chefe[s] := r
3   senão chefe[r] := s
4       se alt[r] = alt[s]
5           alt[s] := alt[r] + 1
```

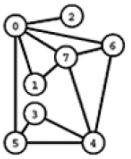


No início do processo, antes que a primeira aresta seja inserida, o altura de cada vértice é 0:

```
INITIALIZE-3 ( )
1   para cada vértice v
2       chefe[v] := v
3       alt[v] := 0
```

A implementação FIND-3 é idêntica à FIND-2.

EXEMPLO A. Vamos executar o algoritmo COMPONENTES para calcular o número de componentes do grafo da figura. Usaremos as implementações FIND-3 e UNION-3 de FIND e UNION. As arestas serão examinadas uma a uma na ordem indicada a seguir.



```
53 71 76 20 07 01 34 54 74 06 46 05
```

Veja o estado dos vetores `chefe` e `alt` no início de cada execução da linha 4 do algoritmo.

```
      chefe[]          alt[]
      0 1 2 3 4 5 6 7  0 1 2 3 4 5 6 7

      0 1 2 3 4 5 6 7  0 0 0 0 0 0 0 0
53    0 1 2 3 4 3 6 7  0 0 0 1 0 0 0 0
71    0 1 2 3 4 3 6 1  0 1 0 1 0 0 0 0
76    0 1 2 3 4 3 1 1  0 1 0 1 0 0 0 0
20    0 1 0 3 4 3 1 1  1 1 0 1 0 0 0 0
07    1 1 0 3 4 3 1 1  1 2 0 1 0 0 0 0
34    1 1 0 3 3 3 1 1  1 2 0 1 0 0 0 0
74    1 1 0 1 3 3 1 1  1 2 0 1 0 0 0 0
```

Veja a evolução da estrutura union-find à medida que arestas são examinadas no grafo. O chefe de cada vértice é indicado por uma traço que “sobe” do vértice ao seu chefe:

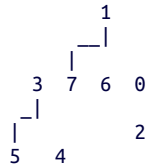
```
          1


    3   7   6   0
```

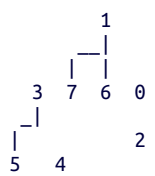
53

3 é o chefe de 5

71

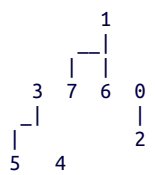


76

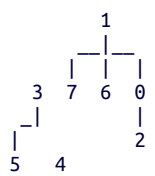


1 é o chefe de 7 e 6

20

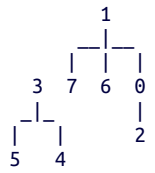


07

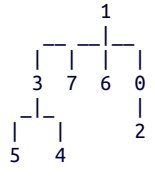


1 é o diretor de 7,6,0,2

01
34



54
74



06
46

```
05
```

Ao final do processo, temos uma só componente e o diretor da componente é 1.

Depois de cada execução de UNION-3, para todo vértice v, $alt[v]$ é a altura de v. Segue daí que, depois de cada operação UNION-3, para cada vértice v, temos

$$alt[v] \le \lg n(v), \qquad \text{(A)}$$

sendo $n(v)$ o número de vértices na subestrutura "pendurada" em v, ou seja, o número de vértices que precisam "passar por" v para chegar ao diretor da componente.

Veja a prova da desigualdade (A): É claro que a desigualdade vale antes da primeira execução da operação UNION-3. Suponha agora que a desigualdade vale antes de alguma execução UNION-3. Vamos mostrar que a desigualdade continua valendo depois da execução.

A execução da operação não diminui $alt[v]$ nem $n(v)$ para nenhum vértice v. Portanto, a única situação que pode ameaçar a validade de (A) é aquela em que o valor de $alt[v]$ aumenta. Mas isso só acontece se $v = s$ e $alt[r] = alt[s]$ (veja a linha 4 de UNION-3). Suponha então que estamos nessa situação e adote as abreviaturas $k = alt[r] = alt[s]$, $M = n(r)$ e $N = n(s)$ no início da operação. Por hipótese de indução, $k \le \lg M$ e $k \le \lg N$, ou seja, $M \ge 2^k$ e $N \ge 2^k$. Depois da operação, teremos

$$\begin{aligned} alt[s] &= k+1 \\ &= \lg 2^{k+1} \\ &= \lg (2^k + 2^k) \\ &\le \lg (M+N) \\ &= \lg n(s) \end{aligned}$$

pois $M+N$ é o valor de $n(s)$ depois da operação (veja a linha 3 de UNION-3). Isso termina a prova de (A).

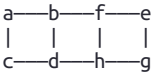
A desigualdade (A) tem a seguinte consequência imediata: no início de cada repetição do bloco de linhas 4-8 de COMPONENTES, a altura de todo vértice é no máximo $\lg n$, sendo n o número de vértices de G. Em particular, a altura de qualquer diretor é no máximo $\lg n$. Como o consumo de tempo de FIND-3 é, no pior caso, proporcional à altura de um diretor, podemos dizer que toda execução de FIND-3 consome

$$O(\lg n)$$

unidades de tempo. Quanto a UNION-3, é claro que essa operação consome O(1) unidades de tempo. Assim, essa terceira implementação do tipo union-find é bem melhor que as anteriores.

Exercícios 4

1. Considere o algoritmo COMPONENTES com as implementações INITIALIZE-3, FIND-3 e UNION-3 de INITIALIZE, FIND e UNION respectivamente. Submeta o



grafo da figura ao algoritmo tomando as arestas na ordem `ba dc db fe hg hf hd ac eg bf`. (Execute o código de UNION-3 *exatamente*.) Dê o estado dos vetores `chefe` e `alt` no fim da execução do algoritmo.

2. É verdade que depois de cada execução da operação UNION-3, para cada vértice v, $alt[v]$ é a profundidade do vértice v?

3. ★ Prove que depois de cada execução da operação UNION-3, para cada vértice v, $alt[v]$ é a altura do vértice v.

4. ★ Complete os detalhes da prova da desigualdade (A).

5. A seguinte variante de UNION-3 está correta?

```
UNION-3A (r, s)
1   se alt[r] = alt[s]
2        chefe[r] := s
3        alt[r] := alt[r] + 1
4   senão se alt[r] > alt[s]
5             chefe[r] := s
6          senão chefe[s] := r
```

6. Mostre que cada execução de FIND-3 consome $O(\lg n)$ unidades de tempo.

7. Quanto tempo o algoritmo COMPONENTES consome para calcular o número de componentes de um grafo com n vértices e m arestas? Suponha que o algoritmo usa as implementações INITIALIZE-3, FIND-3 e UNION-3 de INITIALIZE, FIND e UNION respectivamente. Mostre que COMPONENTES consome $O(n + m \lg n)$ unidades de tempo.

Quarta implementação: path compression

Podemos melhorar o desempenho da terceira implementação da estrutura union-find se usarmos o truque conhecido como *path compression*: a cada execução da operação FIND, plante atalhos no caminho que leva ao diretor da componentes. Mais precisamente, para cada vértice visitado v, adote o diretor da componente de v como novo chefe de v.

```
FIND-4 (v)                ▷ path-compression
1   se v ≠ chefe[v]
2      chefe[v] := FIND-4 (chefe[v])
3   devolva chefe[v]
```

Os códigos de UNION-4 e INITIALIZE-4 são idênticos aos de UNION-3 e INITIALIZE-3 respectivamente.

O consumo de tempo amortizado dessa implementação do union-find é de $O(\log^* n)$ unidades de tempo por operação. Tudo se passa (no sentido amortizado) como se o comprimento de todo caminho na estrutura fosse no máximo $\log^* n$. (Como sempre, n é o número de vértices do grafo.) A função $\log^* n$ cresce muuuuito devagar com n. Portanto, do ponto de vista prático, o desempenho desta implementação é essencialmente igual a $O(1)$.

A prova da cota $O(\log^* n)$ é deveras complexa. (Veja o livro CLRS.)

Exercícios 5

1. Refaça o exemplo A depois de acrescentar a aresta 24 ao grafo. Processe a nova aresta depois de 54 mas antes de 74.

2. Mostre que cada execução de FIND-4 consome $O(\lg n)$ unidades de tempo.

3. ★ Adote as implementações INITIALIZE-4, FIND-4 e UNION-4 da estrutura union-find.
   É verdade que $alt[v] \leq 1$ para todo vértice v?

Veja o verbete Disjoint-set data structure na Wikipedia.

---

www.ime.usp.br/~pf/analise_de_algoritmos/
Atualizado em 2020-11-23

Departamento de Ciência da Computação
Instituto de Matemática e Estatística da USP